EGE322 Iot System Project Project Presentation

Development of Smart Home Automation

Armando ()

Sanjeev (221325c)

Table of contents

01

Introduction

Theme

Preventive Healthcare Measures for Malnutrition

Problem

Despite the increasing awareness of the importance of nutrition, modern lifestyles often lead to imbalanced diets, contributing to malnutrition and related health issues. Traditional methods of diet tracking and management are time-consuming and often lack accuracy, hindering proactive healthcare.

Objective

The primary objective of the smart cabinet is to enhance preventive healthcare measures for malnutrition by providing comprehensive visibility into an individual's dietary habits and health data. This system aims to support better diagnosis and prognosis by offering doctors detailed and accurate information about a patient's daily health and lifestyle. Additionally, it seeks to promote healthy eating habits through personalised recommendations and automated inventory management.

Integration with Healthcare Providers

The smart cabinet's ability to gather and send detailed nutrient consumption reports and health data to doctors bridges the gap between daily habits and medical advice. This integration supports preventive healthcare by enabling doctors to offer more precise recommendations based on accurate, real-time data.

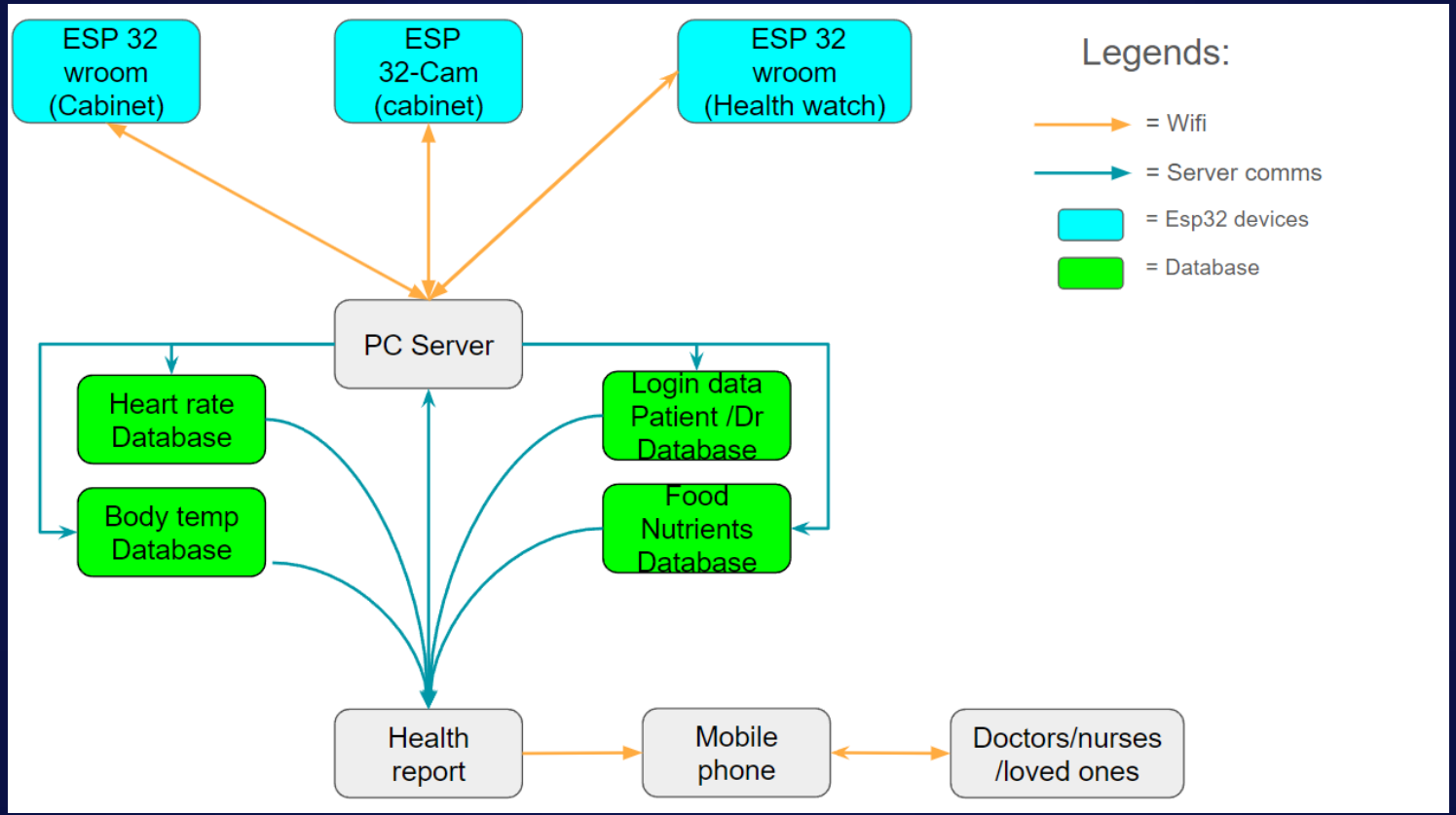
Sub-Objective

- Automate Inventory Management
  - AI-powered Image Recognition: Automatically identifies and catalogs food items.
  - Live Feed and User Interface: Provides a real-time view of the cabinet's contents through an app or web dashboard.
- Impact: Streamlines grocery management, reduces food waste and ensures the family always has fresh and nutritious options.
- Quality of Life (QOL) Changes
  - Promote Healthy Eating: Personalized recipe suggestions and tailored recommendations help the family make healthier food choices.
  - Track and Analyze Eating Habits: Provides insights into dietary patterns, helping users make informed decisions about their diet.
- Impact: Enhances the overall quality of life by making it easier to maintain a healthy diet, ultimately leading to better health outcomes.

02

Overview of Project

System Block Diagram



Project Description

The IoT-enabled smart cabinet is a novel solution designed to address the challenges of maintaining a healthy diet. By combining AI, IoT, and sensor technology, the cabinet provides real-time insights into food consumption, nutritional intake, and overall health. It offers personalized recommendations, automates inventory management, and integrates with healthcare providers to facilitate preventive healthcare.

Project Objective

Enhance preventive healthcare measures for malnutrition through comprehensive dietary monitoring and analysis.

Provide personalized nutrition recommendations to promote healthy eating habits.

Automate inventory management to reduce food waste and ensure access to fresh, nutritious food.

Integrate with healthcare providers to enable better diagnosis and treatment.

Features

1. Habit Tracker

- ESP32 Cam(core): Captures images of food items placed in the cabinet, enabling AI-powered image recognition to identify and catalogue these items.

- Weight Sensor(core): Tracks the amount of food consumed, providing data on eating habits and portion control.

Role: By tracking what and how much food is consumed, the habit tracker helps identify nutritional patterns and deficiencies. This data is crucial for providing personalized health recommendations and alerts.

Features

2. Nutrition Facts and Craving Detector

1. Craving Detector: Analyzes the lack of vitamins in the blood, alerting users to potential nutrient deficiencies.
2. Nutrient Deficit Alert**(core)**: Sends notifications when the diet lacks essential nutrients.

Role: These features ensure that the family maintains a balanced diet, addressing any deficiencies promptly to prevent malnutrition and related health issues.

Features

3. Health Data Gatherer

1. Sensors (Temperature, Humidity, Heart Rate)**(core)**: Collect vital health data from individuals.
2. Logistics: Tracks and logs the use of items from the cabinet, maintaining an accurate inventory.

Role: Gathering comprehensive health data allows for better understanding and monitoring of each family member's health, aiding doctors in making more informed diagnoses and recommendations.

Extra Features

1. Auto Locks : Restricts access to pest and other things allowing the food inside to have a longer shelf life
2. Phone Access and Knock Knock Feature: Allows users to see inside the cabinet via touch or phone, enhancing convenience and preserving the internal environment.

Role: These features ensure that the family follows healthy eating habits and schedules, reducing the risk of overeating or consuming unhealthy snacks.

Our System Uniqueness

UVP

Proactive Health Management

Predicts potential health issues based on your eating habits and provides solutions to prevent them.

Seamless Integration

health data is shared directly with your doctor, allowing for better, more informed care.

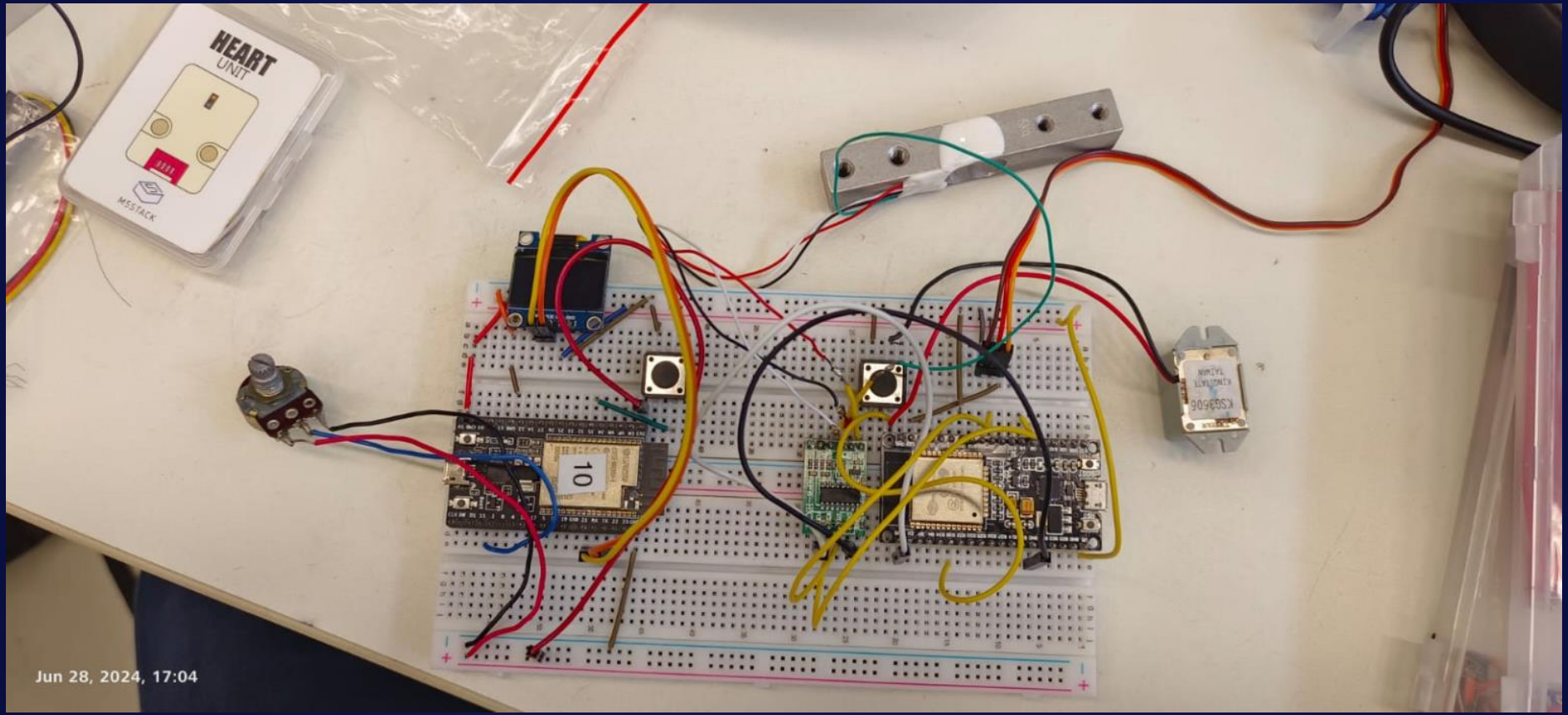
AI-Driven Personalization

The cabinet learns your preferences and adapts to your needs, providing a customized health and nutrition experience.

03

Our prototype

First Proof of concept circuitry



Prototype 1 (Cabinet)

Used a shoebox:

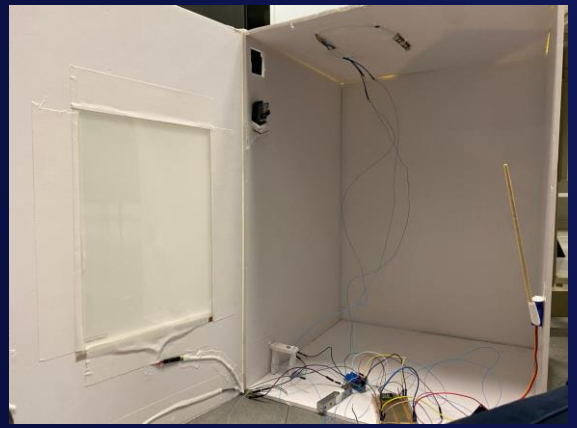
- cut out a hole to mount the esp32 cam
- Initially used UV light strips
- The Light strips are powered by a 12V power supply

Decided against this because:

- space constraint
- visually unappealing

Initial (Cabinet)

Created a mount for the esp32 cam and place it there to take image



- Build the box
- Connected everything to check if the concept works

Improved (Cabinet)

- We realise insufficient light was hitting the esp32 cam we added a led ceiling ring like to the top.
- We mounted the Esp32 cam using hot glue gun after finding the optimal spot for the fov to capture the whole box
- We cleaned up the wiring and did a bit of rewiring and cable management

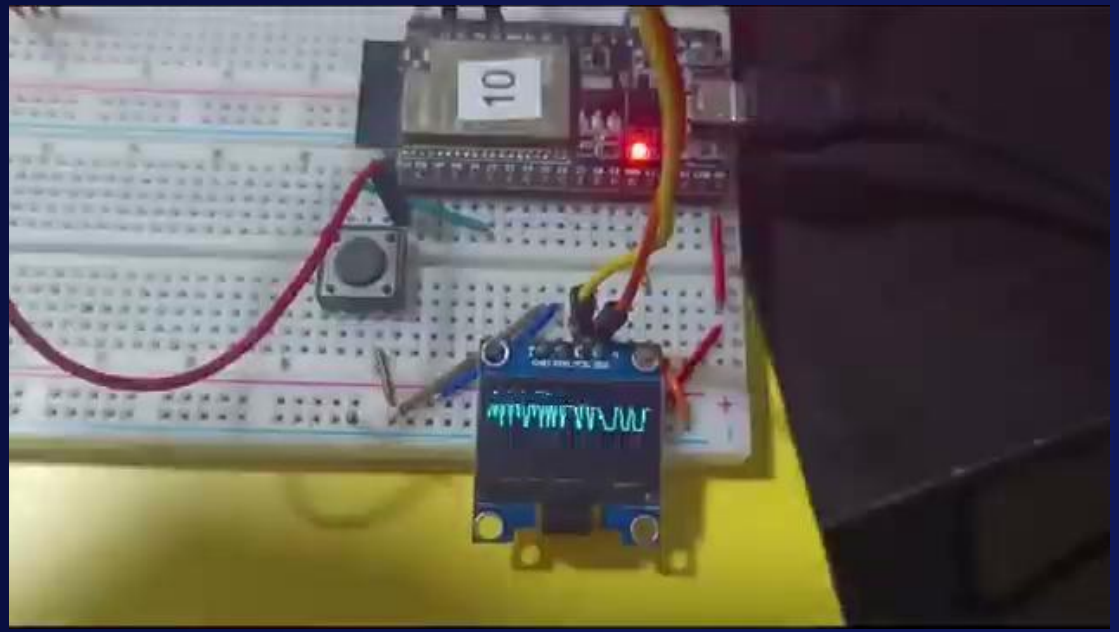
Final (Cabinet)

- We added black foam to reduce the amount of light bouncing into the camera causing it the image to be over exposed
- Created a base for the box so that the components can be hid under and also to allow the load cell to work better

Internal View

External View

smartwatch



This shows the UI of the watch at it initial stages

3d model

3D model made to fit all the components inside to make it a wearable.

04

Hardware Overview

Smart watch Hardware overview

Pulse sensor

The pulse sensor monitors the heart-rate and biofeedback of the body measuring the change in volume of a blood vessel that occur when the heart pumps blood using an optical sensor and green LED.

The HR data is sent to the watch to be displayed in real time and sent which is also sent to the server

Ncir sensor

NCIR is a single-point infrared temperature sensor that detects temperature by measuring the infrared radiation emitted with a temperature measurement range of -70°C to +380°C.

The HR data is sent to the watch to be displayed in real time and sent which is also sent to the server

Oled

The organic light-emitting diode (OLED) display is a mono color, 0.96-inch display with 128×64 pixels

It is used to display all the health data (Heart rate, body temperature) and other health reports and alerts

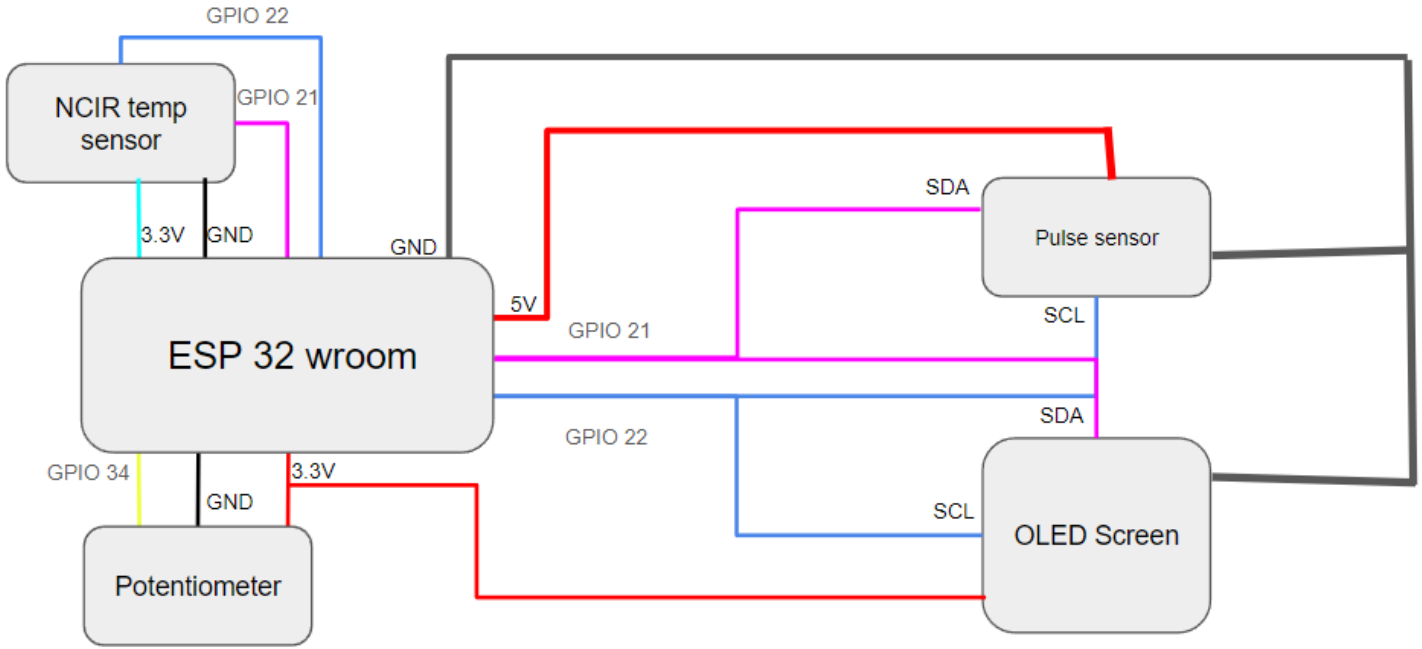
Smart watch Hardware overview

Potentiometer

The potentiometer is used to control the smart watch by allowing it to turn the page and showing hovering over the item it wants to select

Button

The button allows the watch to wake and sleep and also works with the potentiometer to select functions in the watch to display such as the heart rate and the health report



Legends:

= SCL

= SDA

= VCC

= GND

= Analog input

= digital input

Hardware overview (cabinet)

Esp-32 cam

ESP32-CAM is a low-cost ESP32-based development board with onboard camera, small in size

It is used to take image of the inside of the cabinet to send to the server to run it through its yolo ML model to know the items in the cabinet

Relay

The relay is used to turn on the Electrochromic glass in the cabinet so that when the touch sensor is activated it will turn on and let the user look into the cabinet

Led strip

The Led strip is turn on when activated to increase the brightness in the cabinet so that it would be easier for the esp 32 cam to take photos of the items inside the cabinet

Hardware overview (cabinet)

Servo motor

The servo motor will be used to open the door , when prompt it will open or close the locking mechanism allowing the door to be open.

Button

Toggles the servo motor to open and close the door of the cabinet

Electrochromic glass

Electrochromic glass changes its light transmittance only when triggered by an electrical signal

It turns transparent when activated by the touch sensor

Hardware overview (cabinet)

Capacitive touch

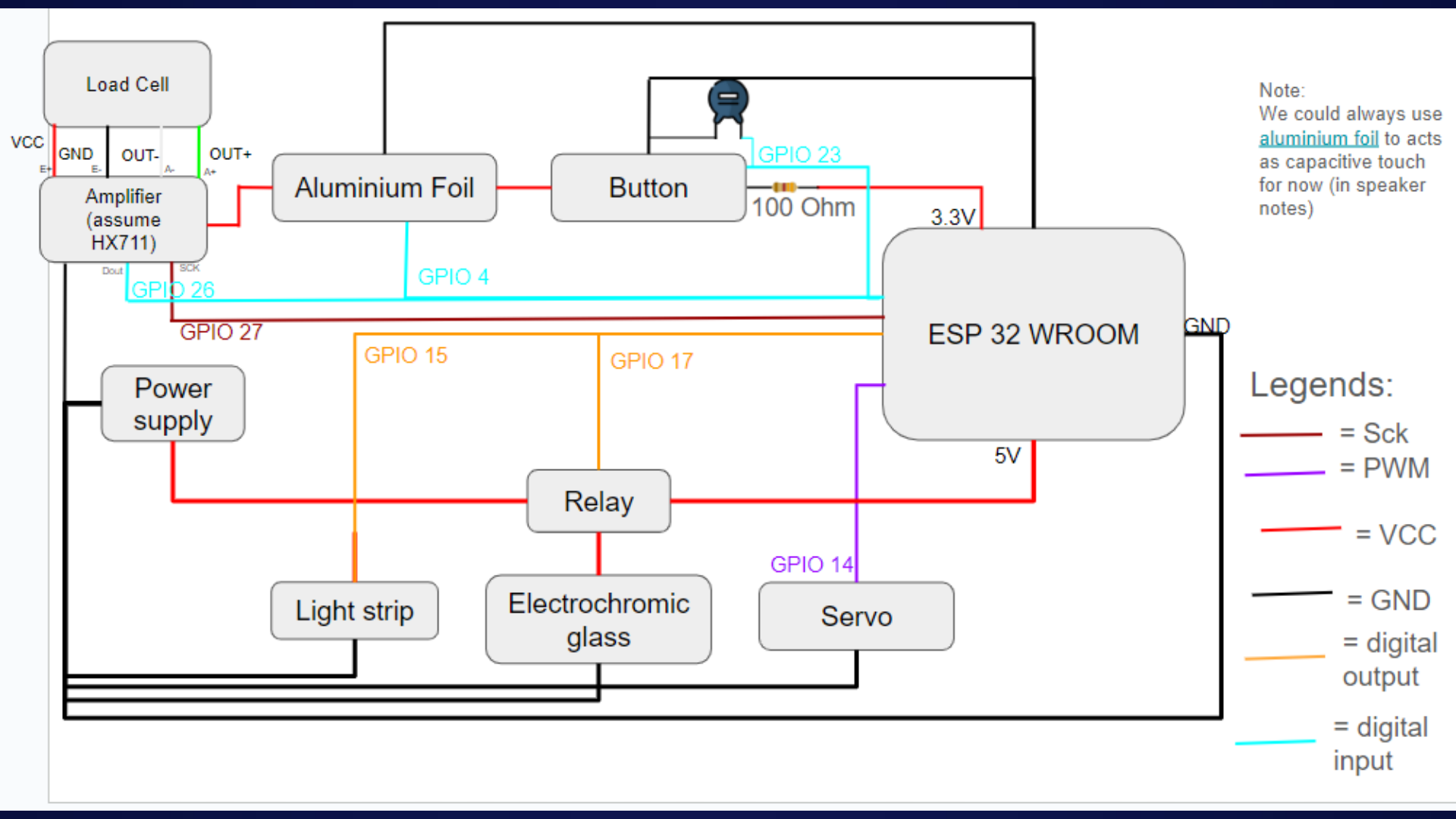
(aluminium foil)

It was self made by us by adding a large piece of aluminium foil where when it is touched it sends a signal to the esp32 to turn the electrochromic glass transparent

Load cell / HX711

The load cell measures mechanical force, mainly the weight of objects. Connected to HX711 a precision 24-bit analog-to-digital converter (ADC).

It is used to track the amount of weight of the items in the box to track the amount of food ate by the user


Load Cell
VCC
GND
OUT-
OUT+
E+
E-
A-
A+
Amplifier (assume HX711)
Dout
SCK
Aluminium Foil
Button
GPIO 23
100 Ohm
3.3V
GPIO 4
GPIO 26
GPIO 27
ESP 32 WROOM
GND
GPIO 15
GPIO 17
Power supply
5V
Relay
GPIO 14
Light strip
Electrochromic glass
Servo
Note:
We could always use aluminium foil to acts as capacitive touch for now (in speaker notes)
Legends:
= Sck
= PWM
= VCC
= GND
= digital output
= digital input

05

Software Overview

Smart health watch

Libraries(OLED)

```python
# MicroPython SSD1306 OLED driver, I2C and SPI interfaces

from micropython import const
import framebuf


# register definitions
SET_CONTRAST = const(0x81)
SET_ENTIRE_ON = const(0xA4)
SET_NORM_INV = const(0xA6)
SET_DISP = const(0xAE)
SET_MEM_ADDR = const(0x20)
SET_COL_ADDR = const(0x21)
SET_PAGE_ADDR = const(0x22)
SET_DISP_START_LINE = const(0x40)
SET_SEG_REMAP = const(0xA0)
SET_MUX_RATIO = const(0xA8)
SET_COM_OUT_DIR = const(0xC0)
SET_DISP_OFFSET = const(0xD3)
SET_COM_PIN_CFG = const(0xDA)
SET_DISP_CLK_DIV = const(0xD5)
SET_PRECHARGE = const(0xD9)
SET_VCOM_DESEL = const(0xDB)
SET_CHARGE_PUMP = const(0x8D)

# Subclassing FrameBuffer provides support for graphics primitives
# http://docs.micropython.org/en/latest/pyboard/library/framebuf.html
class SSD1306(framebuf.FrameBuffer):
    def __init__(self, width, height, external_vcc):
        self.width = width
        self.height = height
        self.external_vcc = external_vcc
        self.pages = self.height // 8
        self.buffer = bytearray(self.pages * self.width)
        super().__init__(self.buffer, self.width, self.height, framebuf.MONO_VLSB)
        self.init_display()

    def init_display(self):
        for cmd in (
            SET_DISP | 0x00,  # off
            # address setting
            SET_MEM_ADDR,
            0x00,  # horizontal
            # resolution and layout
            SET_DISP_START_LINE | 0x00,
            SET_SEG_REMAP | 0x01,  # column addr 127 mapped to SEG0
            SET_MUX_RATIO,
            self.height - 1,
            SET_COM_OUT_DIR | 0x08,  # scan from COM[N] to COM0
            SET_DISP_OFFSET,
            0x00,
            SET_COM_PIN_CFG,
            0x02 if self.width > 2 * self.height else 0x12,
            # timing and driving scheme
            SET_DISP_CLK_DIV,
            0x80,
            SET_PRECHARGE,
            0x22 if self.external_vcc else 0xF1,
            SET_VCOM_DESEL,
            0x30,  # 0.83*Vcc
            # display
            SET_CONTRAST,
            0xFF,  # maximum
            SET_ENTIRE_ON,  # output follows RAM contents
            SET_NORM_INV,  # not inverted
            # charge pump
            SET_CHARGE_PUMP,
            0x10 if self.external_vcc else 0x14,
            SET_DISP | 0x01,
        ):  # on
            self.write_cmd(cmd)
        self.fill(0)
        self.show()

    def poweroff(self):
        self.write_cmd(SET_DISP | 0x00)

    def poweron(self):
        self.write_cmd(SET_DISP | 0x01)

    def contrast(self, contrast):
        self.write_cmd(SET_CONTRAST)
        self.write_cmd(contrast)

    def invert(self, invert):
        self.write_cmd(SET_NORM_INV | (invert & 1))

    def show(self):
        x0 = 0
        x1 = self.width - 1
        if self.width == 64:
            # displays with width of 64 pixels are shifted by 32
            x0 += 32
            x1 += 32
        self.write_cmd(SET_COL_ADDR)
        self.write_cmd(x0)
        self.write_cmd(x1)
        self.write_cmd(SET_PAGE_ADDR)
        self.write_cmd(0)
        self.write_cmd(self.pages - 1)
        self.write_data(self.buffer)


class SSD1306_I2C(SSD1306):
    def __init__(self, width, height, i2c, addr=0x3C, external_vcc=False):
        self.i2c = i2c
        self.addr = addr
        self.temp = bytearray(2)
        self.write_list = [b"\x40", None]  # Co=0, D/C#=1
        super().__init__(width, height, external_vcc)

    def write_cmd(self, cmd):
        self.temp[0] = 0x80  # Co=1, D/C#=0
        self.temp[1] = cmd
        self.i2c.writeto(self.addr, self.temp)

    def write_data(self, buf):
        self.write_list[1] = buf
        self.i2c.writevto(self.addr, self.write_list)


class SSD1306_SPI(SSD1306):
    def __init__(self, width, height, spi, dc, res, cs, external_vcc=False):
        self.rate = 10 * 1024 * 1024
        dc.init(dc.OUT, value=0)
        res.init(res.OUT, value=0)
        cs.init(cs.OUT, value=1)
        self.spi = spi
        self.dc = dc
        self.res = res
        self.cs = cs
        import time

        self.res(1)
        time.sleep_ms(1)
        self.res(0)
        time.sleep_ms(10)
        self.res(1)
        super().__init__(width, height, external_vcc)

    def write_cmd(self, cmd):
        self.spi.init(baudrate=self.rate, polarity=0, phase=0)
        self.cs(1)
        self.dc(0)
        self.cs(0)
        self.spi.write(bytearray([cmd]))
        self.cs(1)

    def write_data(self, buf):
        self.spi.init(baudrate=self.rate, polarity=0, phase=0)
        self.cs(1)
        self.dc(1)
        self.cs(0)
        self.spi.write(buf)
        self.cs(1)
```

A module that I imported so i can run my oled display from the main file

Libraries(HX711)

```python
from utime import sleep_us, time
from machine import Pin
from micropython import const


class HX711Exception(Exception):
    pass


class InvalidMode(HX711Exception):
    pass


class DeviceIsNotReady(HX711Exception):
    pass


class HX711(object):
    """
    Micropython driver for Avia Semiconductor's HX711
    24-Bit Analog-to-Digital Converter
    """
    CHANNEL_A_128 = const(1)
    CHANNEL_A_64 = const(3)
    CHANNEL_B_32 = const(2)

    DATA_BITS = const(24)
    MAX_VALUE = const(0x7fffff)
    MIN_VALUE = const(0x800000)
    READY_TIMEOUT_SEC = const(5)
    SLEEP_DELAY_USEC = const(80)

    def __init__(self, d_out: int, pd_sck: int, channel: int = CHANNEL_A_128):
        self.d_out_pin = Pin(d_out, Pin.IN)
        self.pd_sck_pin = Pin(pd_sck, Pin.OUT, value=0)
        self.channel = channel

    def __repr__(self):
        return "HX711 on channel %s, gain=%s" % self.channel

    def _convert_from_twos_complement(self, value: int) -> int:
        """
        Converts a given integer from the two's complement format.
        """
        if value & (1 << (self.DATA_BITS - 1)):
            value -= 1 << self.DATA_BITS
        return value

    def _set_channel(self):
        """
        Input and gain selection is controlled by the
        number of the input PD_SCK pulses
        3 pulses for Channel A with gain 64
        2 pulses for Channel B with gain 32
        1 pulse for Channel A with gain 128
        """
        for i in range(self._channel):
            self.pd_sck_pin.value(1)
            self.pd_sck_pin.value(0)

    def _wait(self):
        """
        If the HX711 is not ready within READY_TIMEOUT_SEC
        the DeviceIsNotReady exception will be thrown.
        """
        t0 = time()
        while not self.is_ready():
            if time() - t0 > self.READY_TIMEOUT_SEC:
                raise DeviceIsNotReady()

    @property
    def channel(self) -> tuple:
        """
        Get current input channel in a form
        of a tuple (Channel, Gain)
        """
        if self._channel == self.CHANNEL_A_128:
            return 'A', 128
        if self._channel == self.CHANNEL_A_64:
            return 'A', 64
        if self._channel == self.CHANNEL_B_32:
            return 'B', 32
```

```python
    @channel.setter
    def channel(self, value):
        """
        Set input channel
        HX711.CHANNEL_A_128 - Channel A with gain 128
        HX711.CHANNEL_A_64 - Channel A with gain 64
        HX711.CHANNEL_B_32 - Channel B with gain 32
        """
        if value not in (self.CHANNEL_A_128, self.CHANNEL_A_64, self.CHANNEL_B_32):
            raise InvalidMode('Gain should be one of HX711.CHANNEL_A_128, HX711.CHANNEL_A_64, HX711.CHANNEL_B_32')
        else:
            self._channel = value

        if not self.is_ready():
            self._wait()

        for i in range(self.DATA_BITS):
            self.pd_sck_pin.value(1)
            self.pd_sck_pin.value(0)

        self._set_channel()

    def is_ready(self) -> bool:
        """
        When output data is not ready for retrieval,
        digital output pin DOUT is high.
        """
        return self.d_out_pin.value() == 0

    def power_off(self):
        """
        When PD_SCK pin changes from low to high
        and stays at high for longer than 60 us ,
        HX711 enters power down mode.
        """
        self.pd_sck_pin.value(0)
        self.pd_sck_pin.value(1)
        sleep_us(self.SLEEP_DELAY_USEC)

    def power_on(self):
        """
        When PD_SCK returns to low, HX711 will reset
        and enter normal operation mode.
        """
        self.pd_sck_pin.value(0)
        self.channel = self._channel

    def read(self, raw=False):
        """
        Read current value for current channel with current gain.
        If raw is True, the HX711 output will not be converted
        from two's complement format.
        """
        if not self.is_ready():
            self._wait()

        raw_data = 0
        for i in range(self.DATA_BITS):
            self.pd_sck_pin.value(1)
            self.pd_sck_pin.value(0)
            raw_data = raw_data << 1 | self.d_out_pin.value()
        self._set_channel()

        if raw:
            return raw_data
        else:
            return self._convert_from_twos_complement(raw_data)
```

A module that I imported so i can access the data being sent from the load cell to the HX711 to the main file

Libraries(NCIR/MLX90614)

```python
import ustruct

class SensorBase:

    def read16(self, register):
        data = self.i2c.readfrom_mem(self.address, register, 2)
        return ustruct.unpack('<H', data)[0]

    def read_temp(self, register):
        temp = self.read16(register);
        # apply measurement resolution (0.02 degrees per LSB)
        temp *= .02;
        # Kelvin to Celcius
        temp -= 273.15;
        return temp;

    def read_ambient_temp(self):
        return self.read_temp(self._REGISTER_TA)

    def read_object_temp(self):
        return self.read_temp(self._REGISTER_TOBJ1)

    def read_object2_temp(self):
        if self.dual_zone:
            return self.read_temp(self._REGISTER_TOBJ2)
        else:
            raise RuntimeError("Device only has one thermopile")

    @property
    def ambient_temp(self):
        return self.read_ambient_temp()

    @property
    def object_temp(self):
        return self.read_object_temp()

    @property
    def object2_temp(self):
        return self.read_object2_temp()

class MLX90614(SensorBase):

    _REGISTER_TA = 0x06
    _REGISTER_TOBJ1 = 0x07
    _REGISTER_TOBJ2 = 0x08

    def __init__(self, i2c, address=0x5a):
        self.i2c = i2c
        self.address = address
        _config1 = i2c.readfrom_mem(address, 0x25, 2)
        _dz = ustruct.unpack('<H', _config1)[0] & (1<<6)
        self.dual_zone = True if _dz else False

class MLX90615(SensorBase):

    _REGISTER_TA = 0x26
    _REGISTER_TOBJ1 = 0x27

    def __init__(self, i2c, address=0x5b):
        self.i2c = i2c
        self.address = address
        self.dual_zone = False
```

A module that I imported so i can access the data being sent by the NCIR sensor from the main file

TEMP(Temperature)

```
from machine import Pin, PWM, I2C, Timer, ADC
from time import sleep
import ssd1306
import HR
import TEMP
import framebuf




temp_data = [
                0b00000011, 0b00000000,
                0b00000100, 0b10011000,
                0b00000100, 0b10000000,
                0b00000100, 0b10011100,
                0b00000100, 0b10000000,
                0b00000100, 0b10010000,
                0b00000100, 0b10000000,
                0b00000100, 0b10011100,
                0b00000100, 0b10000000,
                0b00000100, 0b10011000,
                0b00000100, 0b10000000,
                0b00001000, 0b01000000,
                0b00010000, 0b00100000,
                0b00100000, 0b00010000,
                0b00100000, 0b00010000,
                0b00100000, 0b00010000,
                0b00100000, 0b00010000,
                0b00010000, 0b00100000,
                0b00001000, 0b01000000,
                0b00000111, 0b10000000,


    ]

def TEMP(display, button_pin, temp):
            display.fill(0)
            buffer = bytearray(temp_data)
            fb = framebuf.FrameBuffer(buffer, 16, 20, framebuf.MONO_HLSB)
            display.blit(fb, 49, 17)  # Adjust coordinates as needed for centering
            temperature = temp.read_object_temp()
            display.text("{:.2f}C".format(temperature), 52, 47, 1)
            display.show()
            sleep(1)
```

- **Initialization:**
    a. Imports libraries.
    b. Sets up pins for button, display communication (likely I2C), etc. (not shown).
- **Heart Rate:**
    a. HR module likely handles pulse sensor reading, peak detection, and BPM calculation (details not shown here).
- **Temperature:**
    a. TEMP module likely reads the temperature sensor.
    b. TEMP function displays a custom temperature icon on the OLED using `framebuf`.
    c. It then reads the temperature and displays it with two decimal places.

Display Updates:

- The code likely calls functions from HR and TEMP to retrieve heart rate and temperature values.
- It refreshes the display with the heart rate (potentially using a separate function) and temperature using the TEMP function shown.

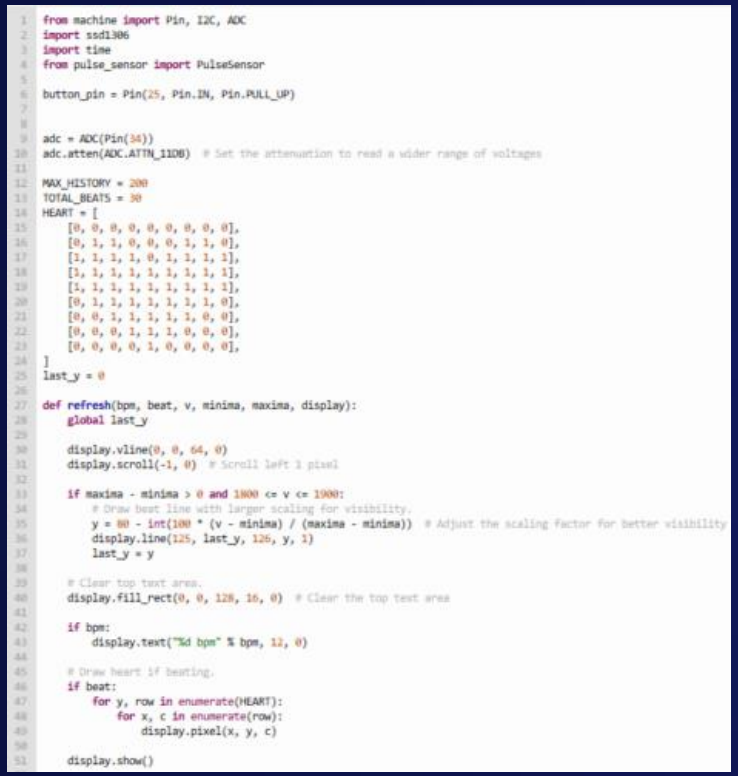
Libraries(Pulse sensor)

```
from machine import Pin, ADC
from time import sleep, ticks_ms, ticks_diff
import time

class PulseSensor:
    def __init__(self, adc_pin=34, led_pin=2, low_threshold=1800, high_threshold=1900, smoothing_window_size=5, bpm_estimation_interval=15):
        self.adc = ADC(Pin(adc_pin))
        self.adc.atten(ADC.ATTN_11DB)
        self.led = Pin(led_pin, Pin.OUT)

        # Threshold values
        self.low_threshold = low_threshold
        self.high_threshold = high_threshold

        # Peak detection variables
        self.previous_signal = 0
        self.peak_detected = False
        self.last_peak_time = 0

        # BPM calculation variables
        self.peak_times = []
        self.bpm_estimation_interval = bpm_estimation_interval  #
        self.peak_count = 0
        self.start_time = time.ticks_ms()

        # Kalman filter variables
        self.Q = 0.1  # Process noise covariance
        self.R = 0.1  # Measurement noise covariance
        self.X = 0.0  # Value
        self.P = 1.0  # Estimation error covariance
        self.K = 0.0  # Kalman gain

        # Smoothing variables
        self.smoothing_window_size = smoothing_window_size
        self.signal_window = []

        # Peak count variables for 15-second interval BPM calculation
        self.peak_count = 0
        self.start_time = time.ticks_ms()

        # Method switch
        self.use_15sec_method = False

    def kalman_filter(self, value):
        self.P = self.P + self.Q
        self.K = self.P / (self.P + self.R)
        self.X = self.X + self.K * (value - self.X)
        self.P = (1 - self.K) * self.P
        return self.X

    def smooth_signal(self, signal):
        self.signal_window.append(signal)
        if len(self.signal_window) > self.smoothing_window_size:
            self.signal_window.pop(0)
        return sum(self.signal_window) / len(self.signal_window)

    def read_sensor(self):
        return self.adc.read()

    def detect_peaks_and_calculate_bpm(self):
        current_signal = self.read_sensor()  # Read the PulseSensor's value
        smoothed_signal = self.smooth_signal(current_signal)  # Smooth the signal
        filtered_signal = self.kalman_filter(smoothed_signal)  # Apply Kalman filter

        estimated_bpm = 0

        # Check if the signal is within the desired range
        if self.low_threshold < filtered_signal < self.high_threshold:
            if filtered_signal > self.previous_signal:
                self.peak_detected = True
            elif filtered_signal < self.previous_signal and self.peak_detected:
                # A peak is detected
                current_time = time.ticks_ms()
                if time.ticks_diff(current_time, self.last_peak_time) > 300:  # Minimum interval to avoid noise
                    self.last_peak_time = current_time
                    self.peak_times.append(current_time)

                    # Remove old peaks outside the estimation window
                    self.peak_times = [t for t in self.peak_times if time.ticks_diff(current_time, t) <= self.bpm_estimation_interval * 1000]

                    # Calculate BPM if there are at least two peaks
                    if len(self.peak_times) > 1:
                        intervals = [time.ticks_diff(self.peak_times[i], self.peak_times[i - 1]) for i in range(1, len(self.peak_times))]
                        avg_interval = sum(intervals) / len(intervals)
                        estimated_bpm = 60000 / avg_interval
```

```
                self.led.on()  # Turn on LED for a heartbeat
                print(f"Heartbeat detected, estimated BPM: {estimated_bpm:.2f}")
            self.peak_detected = False
        else:
            self.led.off()  # Turn off LED if no heartbeat is detected
    else:
        self.peak_detected = False
        self.led.off()  # Turn off LED if signal is out of range

    # Update previous signal
    self.previous_signal = filtered_signal

    return estimated_bpm

# Inside PulseSensor class

def calculate_bpm_over_15sec(self):
    self.start_time = time.ticks_ms()
    self.peak_count = 0
    while time.ticks_diff(time.ticks_ms(), self.start_time) < 15000:
        current_signal = self.read_sensor()  # Read the PulseSensor's value
        smoothed_signal = self.smooth_signal(current_signal)  # Smooth the signal
        filtered_signal = self.kalman_filter(smoothed_signal)  # Apply Kalman filter


        # Check if the signal is within the desired range
        if self.low_threshold < filtered_signal < self.high_threshold:
            if filtered_signal > self.previous_signal:
                self.peak_detected = True
            elif filtered_signal < self.previous_signal and self.peak_detected:
                # A peak is detected
                self.peak_detected = False
                self.peak_count += 1
                self.led.on()  # Turn on LED for a heartbeat
                print("Heartbeat detected")
            else:
                self.led.off()  # Turn off LED if no heartbeat is detected
        else:
            self.peak_detected = False
            self.led.off()  # Turn off LED if signal is out of range

        # Update previous signal
        self.previous_signal = filtered_signal
        print(filtered_signal)

        # Delay for stability
        time.sleep_ms(50)

    # Calculate BPM after 15 seconds
    bpm = self.peak_count * 4  # Calculate BPM based on peaks counted
    print(f"BPM: {bpm}")

    return bpm
```

This is a Custom made library, made by us to get the data from the pulse sensor and access it from the main file

HR (Heart rate)



```python
from machine import Pin, I2C, ADC
import ssd1306
import time
from pulse_sensor import PulseSensor

button_pin = Pin(25, Pin.IN, Pin.PULL_UP)


adc = ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)  # Set the attenuation to read a wider range of voltages

MAX_HISTORY = 200
TOTAL_BEATS = 30
HEART = [
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 1, 1, 0, 0, 0, 1, 0],
    [1, 1, 1, 1, 0, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1],
    [0, 1, 1, 1, 1, 1, 1, 0],
    [0, 0, 1, 1, 1, 1, 0, 0],
    [0, 0, 0, 1, 1, 0, 0, 0],
    [0, 0, 0, 0, 1, 0, 0, 0],
]
last_y = 0

def refresh(bpm, beat, v, minima, maxima, display):
    global last_y

    display.vline(0, 0, 64, 0)
    display.scroll(-1, 0)  # Scroll left 1 pixel

    if maxima - minima > 0 and 1000 <= v <= 1900:
        # Draw beat line with larger scaling for visibility
        y = 60 - int(100 * (v - minima) / (maxima - minima))  # Adjust the scaling factor for better visibility
        display.line(125, last_y, 126, y, 1)
        last_y = y

    # Clear top text area
    display.fill_rect(0, 0, 128, 16, 0)  # Clear the top text area

    if bpm:
        display.text("%d bpm" % bpm, 12, 0)

    # Draw heart if beating
    if beat:
        for y, row in enumerate(HEART):
            for x, c in enumerate(row):
                display.pixel(x, y, c)

    display.show()
```

1. **Pin Configuration:**
   - `button_pin`: Input pin for the button with pull-up resistor (active when not pressed).
   - `adc`: Instance for analog-to-digital conversion (reading sensor voltage).
   - `adc.atten(ADC.ATTN_11DB)`: Adjusts voltage reading range for better sensor compatibility.
2. **Constants:**
   - `MAX_HISTORY`: Maximum number of recent sensor readings to store (e.g., 200).
   - `TOTAL_BEATS`: Maximum number of recent heartbeats to track (e.g., 30).
3. **Heartbeat Visualization (HEART):**
   - Predefined pattern representing a heartbeat displayed on the screen.
4. **`refresh` Function (Display Update):**
   - Clears the display area and scrolls content left for animation.
   - **Heartbeat Line (if detected):**
     - Calculates a scaled position based on voltage for visualization.
     - Draws a line representing the heartbeat.
   - **Heart Rate Text:** Displays calculated heart rate if available.
   - **Heartbeat Animation (if detected):**
     - Iterates through the `HEART` pattern to draw pixels and create the animation.
   - Updates the display with all drawn elements.

HR (Heart rate)

```python
def HR(display,button_pin):
    history = []
    beats = []
    beat = False
    bpm = None
    last_valid_bpm = 60

    Hr_sensor = PulseSensor()

    while True:
        button_state = button_pin.value()
        if button_state == 1:
            bpm = Hr_sensor.detect_peaks_and_calculate_bpm()
            v = Hr_sensor.read_sensor()
            raw_data = adc.read()
            history.append(raw_data)

            # Get the tail, up to MAX_HISTORY length
            history = history[-MAX_HISTORY:]

            minima, maxima = 1700, 2000

            if bpm > 0:
                beat = True
                beats.append(time.time())
                # Truncate beats queue to max
                beats = beats[-TOTAL_BEATS:]
                last_valid_bpm = bpm  # Update last valid BPM
            else:
                beat = False
                bpm = last_valid_bpm  # Use last valid BPM if current BPM is zero

            refresh(bpm, beat, v, minima, maxima, display)

        elif button_state == 0:
            break
```

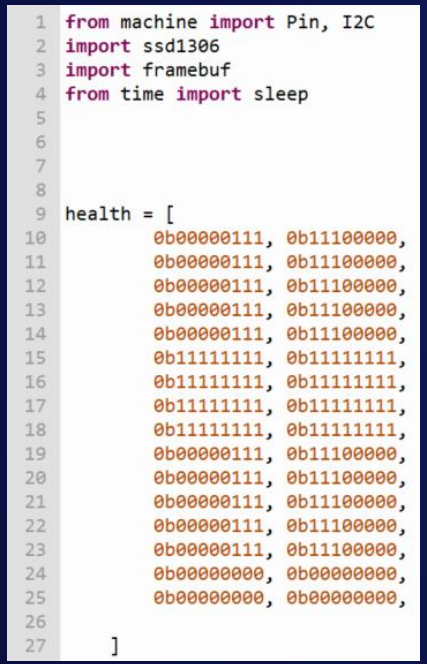
The code is designed to measure and display heart rate using a pulse sensor.

Here's how it works:

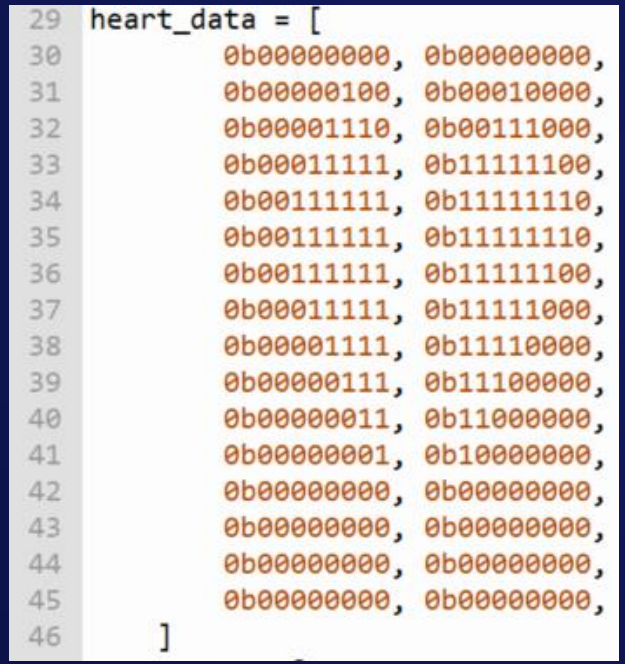
1. **Data Collection:** It continuously reads data from the pulse sensor and stores recent readings.
2. **Heartbeat Detection:** It analyzes the sensor data to identify peaks, which represent heartbeats.
3. **Heart Rate Calculation:** It calculates the heart rate based on the time between detected heartbeats.
4. **Display Update:** It displays the calculated heart rate and other relevant information on a connected display.
5. **Button Control:** The code waits for a button press to start or stop the heart rate measurement.

Page_1 (Homepage design)

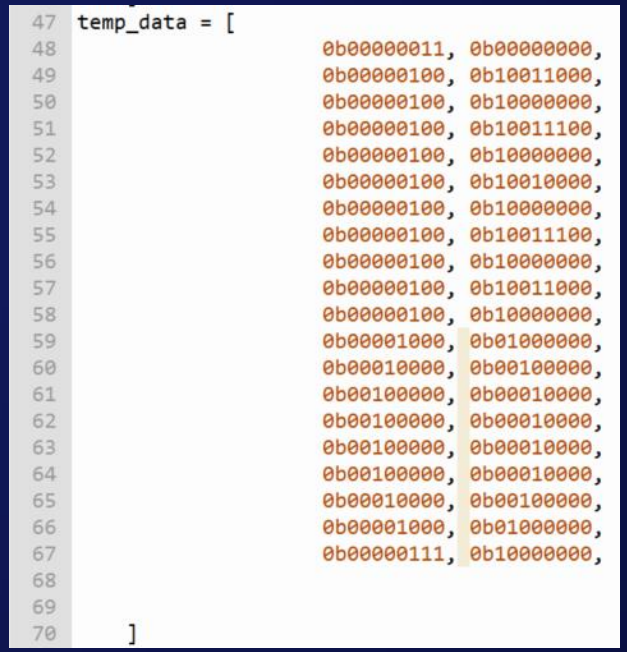
This are the designs for the heart, thermometer, and health report



```
from machine import Pin, I2C
import ssd1306
import framebuf
from time import sleep



health = [
        0b00000111, 0b11100000,
        0b00000111, 0b11100000,
        0b00000111, 0b11100000,
        0b00000111, 0b11100000,
        0b00000111, 0b11100000,
        0b11111111, 0b11111111,
        0b11111111, 0b11111111,
        0b11111111, 0b11111111,
        0b11111111, 0b11111111,
        0b00000111, 0b11100000,
        0b00000111, 0b11100000,
        0b00000111, 0b11100000,
        0b00000111, 0b11100000,
        0b00000111, 0b11100000,
        0b00000000, 0b00000000,
        0b00000000, 0b00000000,

    ]
```



```
heart_data = [
        0b00000000, 0b00000000,
        0b00000100, 0b00010000,
        0b00001110, 0b00111000,
        0b00011111, 0b11111100,
        0b00111111, 0b11111110,
        0b00111111, 0b11111110,
        0b00111111, 0b11111100,
        0b00011111, 0b11111000,
        0b00001111, 0b11110000,
        0b00000111, 0b11100000,
        0b00000011, 0b11000000,
        0b00000001, 0b10000000,
        0b00000000, 0b00000000,
        0b00000000, 0b00000000,
        0b00000000, 0b00000000,
        0b00000000, 0b00000000,
    ]
```



```
temp_data = [
                    0b00000011, 0b00000000,
                    0b00000100, 0b10011000,
                    0b00000100, 0b10000000,
                    0b00000100, 0b10011100,
                    0b00000100, 0b10000000,
                    0b00000100, 0b10010000,
                    0b00000100, 0b10000000,
                    0b00000100, 0b10011100,
                    0b00000100, 0b10000000,
                    0b00000100, 0b10011000,
                    0b00000100, 0b10000000,
                    0b00001000, 0b01000000,
                    0b00010000, 0b00100000,
                    0b00100000, 0b00010000,
                    0b00100000, 0b00010000,
                    0b00100000, 0b00010000,
                    0b00100000, 0b00010000,
                    0b00010000, 0b00100000,
                    0b00001000, 0b01000000,
                    0b00000111, 0b10000000,


    ]
```

Page_1 (Homepage design)

```python
def select(rows,column,display):
    grid_width = 2
    grid_height = 2
    cell_size = 30
    cell_spacing = 1
    selected_row = rows    # Initially select the top-left cell
    selected_col = column

    # Calculate starting position
    start_x = (display.width // 2) - ((grid_width * cell_size + (grid_width - 1) * cell_spacing) // 2)
    start_y = (display.height // 2) - ((grid_height * cell_size + (grid_height - 1) * cell_spacing) // 2)

    while True:
        display.fill(0)  # Clear the screen

        # Draw the Grid
        for row in range(grid_height):
            for col in range(grid_width):
                x = start_x + col * (cell_size + cell_spacing)
                y = start_y + row * (cell_size + cell_spacing)
                if row == selected_row and col == selected_col:
                    display.fill_rect(x, y, cell_size, cell_size, 1)  # Filled for selected cell
                else:
                    display.rect(x, y, cell_size, cell_size, 1)  # Outline for other cells
        HR = bytearray(heart_data)
        TEMP = bytearray(temp_data)
        Health = bytearray(health)
        fbHR = framebuf.FrameBuffer(HR, 16, 16, framebuf.MONO_HLSB)
        fbTEMP = framebuf.FrameBuffer(TEMP, 16, 20, framebuf.MONO_HLSB)
        fbHealth = framebuf.FrameBuffer(Health, 16, 16, framebuf.MONO_HLSB)
        display.blit(fbHR, 42,12)
        display.blit(fbTEMP, 75,8)
        display.blit(fbHealth, 42,41)
        display.show()
        break
```

Functionality: Presents a grid layout on the OLED display and allows selection of a cell using button presses (button handling likely not shown).
Grid Parameters:

- Number of rows and columns (`rows`, `column` - likely received as arguments)
- Cell size (`cell_size`)
- Spacing between cells (`cell_spacing`)

Display Update Loop:

1. Clears the screen.
2. Draws the grid:
    - Iterates through rows and columns.
    - Calculates the position of each cell based on screen size and grid parameters.
    - Draws a filled rectangle for the currently selected cell.
    - Otherwise, draws an outline for other cells.
3. Displays health icons (assuming `heart_data`, `temp_data`, and `health` are predefined byte arrays representing icon data):
    - Converts each bytearray to a `framebuf.FrameBuffer` object for efficient display.
    - "Blitz" (draws) each icon buffer onto the OLED display at specific coordinates.
4. Shows the updated display.

Main code (Initialisation/setup)

```
13  # Potentiometer setup
14  pot_pin = ADC(Pin(33))
15  pot_pin.atten(ADC.ATTN_11DB)
16  pot_pin.width(ADC.WIDTH_12BIT)
17
18  # Button setup
19  button_pin = Pin(25, Pin.IN, Pin.PULL_UP)
20
21  # OLED setup
22  i2c = I2C(sda=Pin(21), scl=Pin(22), freq=100000)
23  display = ssd1306.SSD1306_I2C(128, 64, i2c)
24  temp_sensor = mlx.MLX90614(i2c)
25
26  # Pulse Sensor setup
27  Hr_sensor = PulseSensor()
28
29  temp_sensor = mlx.MLX90614(i2c)
```

Potentiometer:

- Creates an ADC object (`pot_pin`) for reading analog values from pin 33.
- Sets the voltage attenuation to 11dB (`ADC.ATTN_11DB`) for better resolution at lower voltages.
- Sets the analog-to-digital conversion width to 12 bits (`ADC.WIDTH_12BIT`) for higher precision.

Button:

- Creates a Pin object (`button_pin`) for reading button state from pin 25.
- Sets the pin mode to input (`Pin.IN`) and enables a pull-up resistor (`Pin.PULL_UP`) to ensure a high default state.

OLED:

- Creates an I2C object (`i2c`) for communicating with the OLED display.
- Creates an SSD1306 object (`display`) to control the OLED display with a resolution of 128x64 pixels.
- Creates an MLX90614 object (`temp_sensor`) for reading temperature using the MLX90614 sensor.

Pulse Sensor:

- Creates a PulseSensor object (`Hr_sensor`) for measuring heart rate from the pulse sensor (implementation details might vary depending on the library).

Main code (WifiConect module)

```
31  # WLAN setup
32  sta_if = network.WLAN(network.STA_IF)
33  sta_if.active(True)
34  WIFI_SSID = "MFDAB Wifi"
35  WIFI_PASS = "SANJEEV1303"
36
37  # Socket Configuration
38  SERVER_IP = "192.168.18.2"  # Server IP
39  SERVER_PORT = 5001  # Server Port
40  s = None  # Socket object
41
42  def connect_wifi():
43      sta_if.connect(WIFI_SSID, WIFI_PASS)
44      while not sta_if.isconnected():
45          sleep(1)
46      print('Connected to Wi-Fi with IP:', sta_if.ifconfig()[0])
```

This function establishes a Wi-Fi connection to the specified network.

1. WLAN setup would activate the Wi-Fi interface (`sta_if`) and set the SSID and password for connection.
2. Socket Configuration initializes variables for the server IP address, server port, and the socket object (`s`) which starts as `None` (not yet connected).
3. **Connects to the Wi-Fi network:** Uses the `sta_if.connect()` method to initiate the connection using the provided SSID and password.
4. **Waits for connection:** Continuously checks if the connection is established using `sta_if.isconnected()`. If not, it waits for 1 second before checking again.
5. **Prints connection status:** Once connected, it prints the device's IP address to the console for confirmation.

Main code (Server Communications)

```
49  def server_comms(s):
50      while True:
51          try:
52              data = s.recv(1024)  # <8 = ok
53              if data == b"bpm":
54                  bpm_value = Hr_sensor.calculate_bpm_over_15sec()
55                  temperature = temp_sensor.read_object_temp()  # Correct way to
56                  s.send(f"{bpm_value:.2f},bpm".encode("utf-8"))
57                  s.send(f"{temperature:.2f},temp".encode("utf-8"))
58          except OSError as e:
59              print("Socket error:", e)
60              # Handle the error as needed, e.g., reconnect or continue
```

This function handles communication with the server.

1. **Continuous loop:** Runs indefinitely to maintain a connection.
2. **Receives data:** Attempts to receive data from the server using `s.recv(1024)`.
3. **Checks for "bpm" command:** If the received data is "bpm", it calculates the heart rate and temperature.
4. **Sends data to server:** Sends the calculated heart rate and temperature values to the server in a specific format.
5. **Error handling:** If a socket error occurs, it prints an error message and likely handles the error (e.g., reconnecting).

Main code (Main loop)

```
def main_loop():
    while True:
        pot_value = pot_pin.read()
        button_state = button_pin.value()

        if pot_value == 0:
            page_1.select(3, 3)

        elif pot_value <= 1024:
            page_1.select(0, 0)
            if button_state == 0:
                sleep(0.5)
                HR.HR(display, button_pin)
                page_1.select(3, 3)
                sleep(1)

        elif pot_value <= 2048:
            page_1.select(1, 0)

        elif pot_value <= 3072:
            page_1.select(0, 1)
            if button_state == 0:
                sleep(0.2)
                while True:
                    button_state = button_pin.value()
                    if button_state == 1:
                        sleep(0.5)
                        TEMP.TEMP(display, button_pin, temp)
                        continue
                    elif button_state == 0:
                        break
                page_1.select(3, 3)
                sleep(1)

        elif pot_value <= 4096:
            page_1.select(1, 1)
```

This function is the main program loop that controls the device's behavior.

1. **Continuous loop:** Runs indefinitely to keep the program running.
2. **Reads potentiometer and button values:** Reads the current values from the potentiometer and button.
3. **Handles potentiometer value ranges:** Based on the potentiometer value, it selects what information should be displayed on the OLED.
4. **Button handling:** If the button is pressed, it performs specific actions like starting heart rate measurement or displaying temperature.

Main code (Main Program)

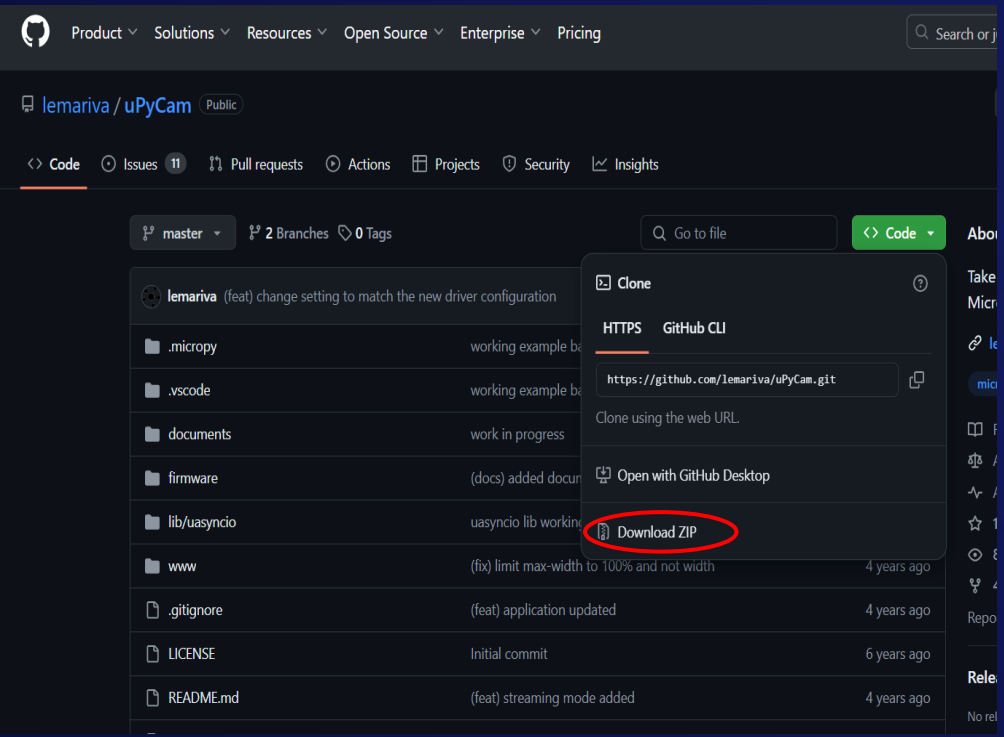
```
 99  # Start main loop and server communications
100  connect_wifi()
101  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
102  s.connect((SERVER_IP, SERVER_PORT))
103  print('Connected to server')
104  s.send(b"new~wch")
105
106  _thread.start_new_thread(server_comms, (s,))
107  main_loop()
```

- Establishes Wi-Fi connection.
- Creates a TCP socket for server communication.
- Connects to the specified server.
- Sends an initial message to the server.
- Starts a new thread to handle server communication.
- Initiates the main program loop.

Cabinet

How to install Lemivara's firmware for esp32 cam (option 1)

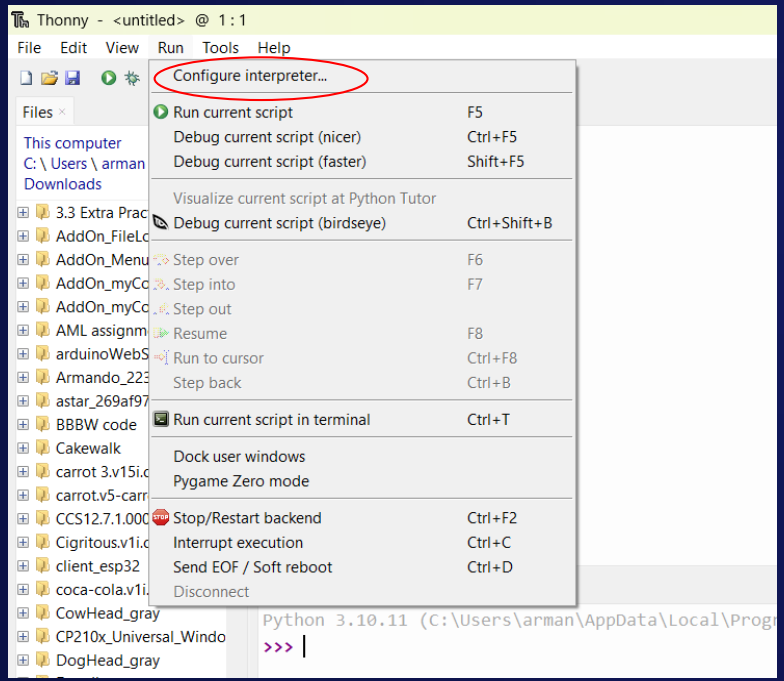
1. Download the whole code



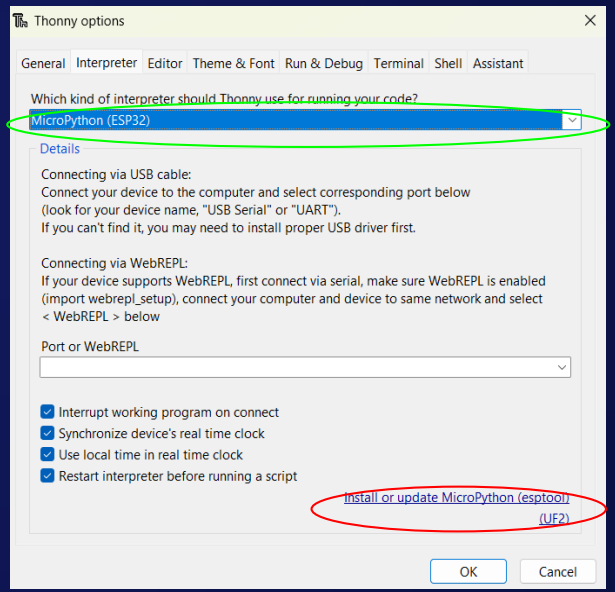
2. Unzip the folder

How to install Lemivara's firmware for esp32 cam (option 1)

3. Go to "Run" -> "Configure Interpreter.."

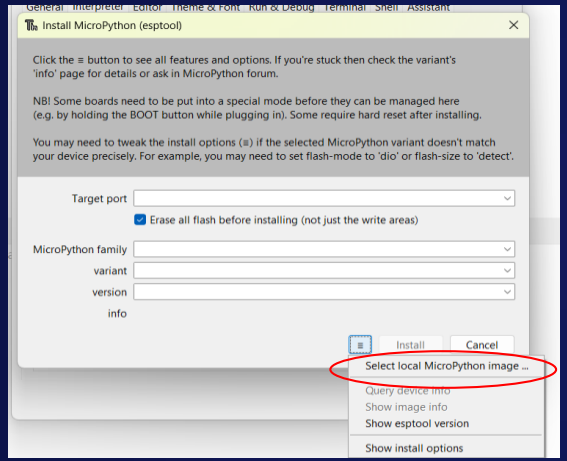


4. Select "Micropython (ESP32)" (green) and follow below

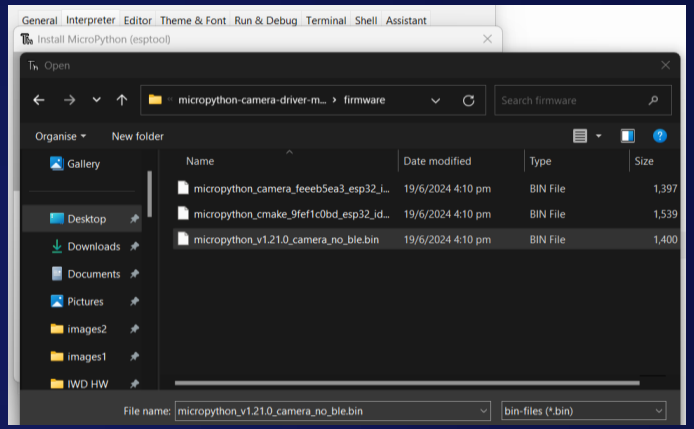


How to install Lemivara's firmware for esp32 cam (option 1)

5. Find the  icon and follow below

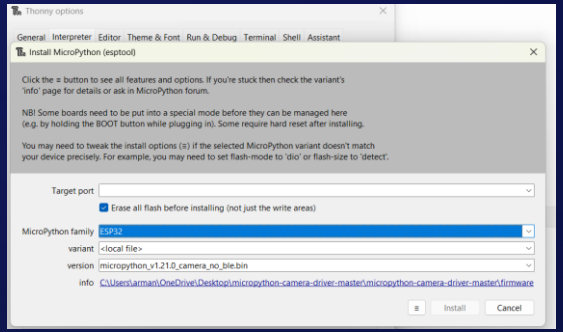


6. Find the file under "firmware"



Note:
Some people could use the firmware with the "_cmake_". However, I only was successful in using the "camera_no_ble"

7. Press install



To check if your firmware is working check it with your serial monitor

- any will do: Arduino IDE(assuming you have download the esp32 board driver before) , Thonny or even putty (if you can configure)

How to install Lemivara's firmware for esp32 cam (option 2)

```
esptool.py --chip esp32 --port /dev/ttyUSB0 erase_flash
esptool.py --chip esp32 --port /dev/ttyUSB0 write_flash -z 0x1000 micropython_camera_feeeb5ea3_esp32_id
```

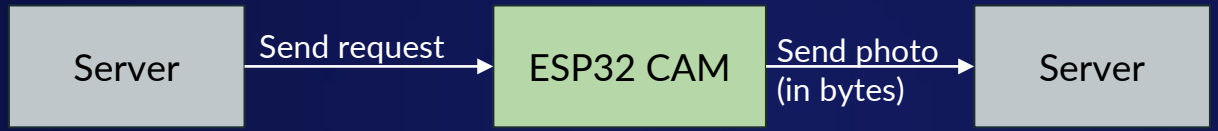
- Note: if this code run on your command prompt and gives you an error, you could always remove the "--port /dev/ttyUSB0" or "--port COM7"
- To check if your firmware is working check it with your serial monitor
  - any will do: Arduino IDE(assuming you have download the esp32 board driver before) , Thonny or even putty (if you can configure)

Context

Server always prompt the cabinet to do a task. The response to this could be:

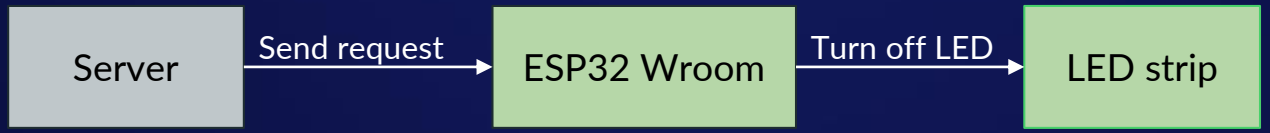
- **Gather and send sensor information**
  - E.g. Request for ESP32 Cam to take pictures

Sequence of events:



- **Actuate an actuator**
  - E.g. Turns off the LED strips

Sequence of events:

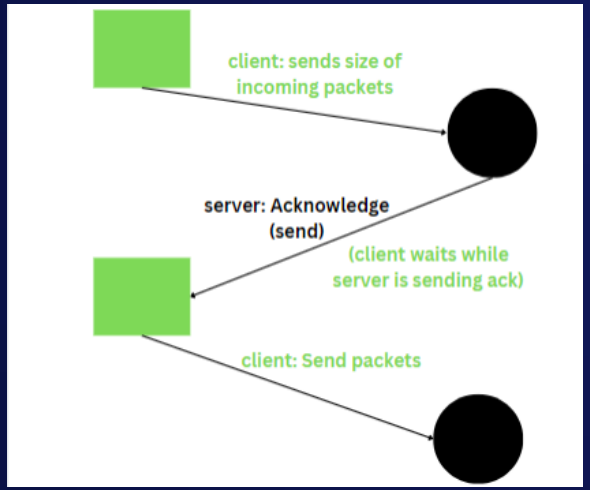


All devices here will follow the convention 1 of sending info from ESP32 to server

Sending info using sockets

Convention 1

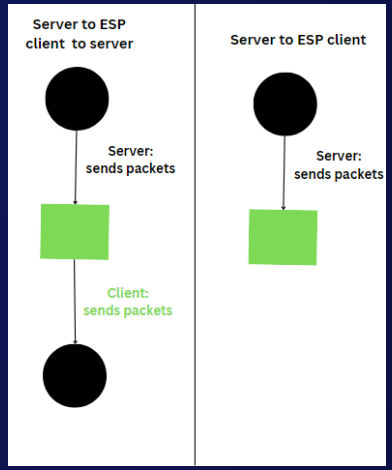
- Before sending message:
  - Send the size of the message to be sent
- After acknowledgement for size declaration:
  - Send the message to receiver



- Only seen by Client to server (more details in assignment)

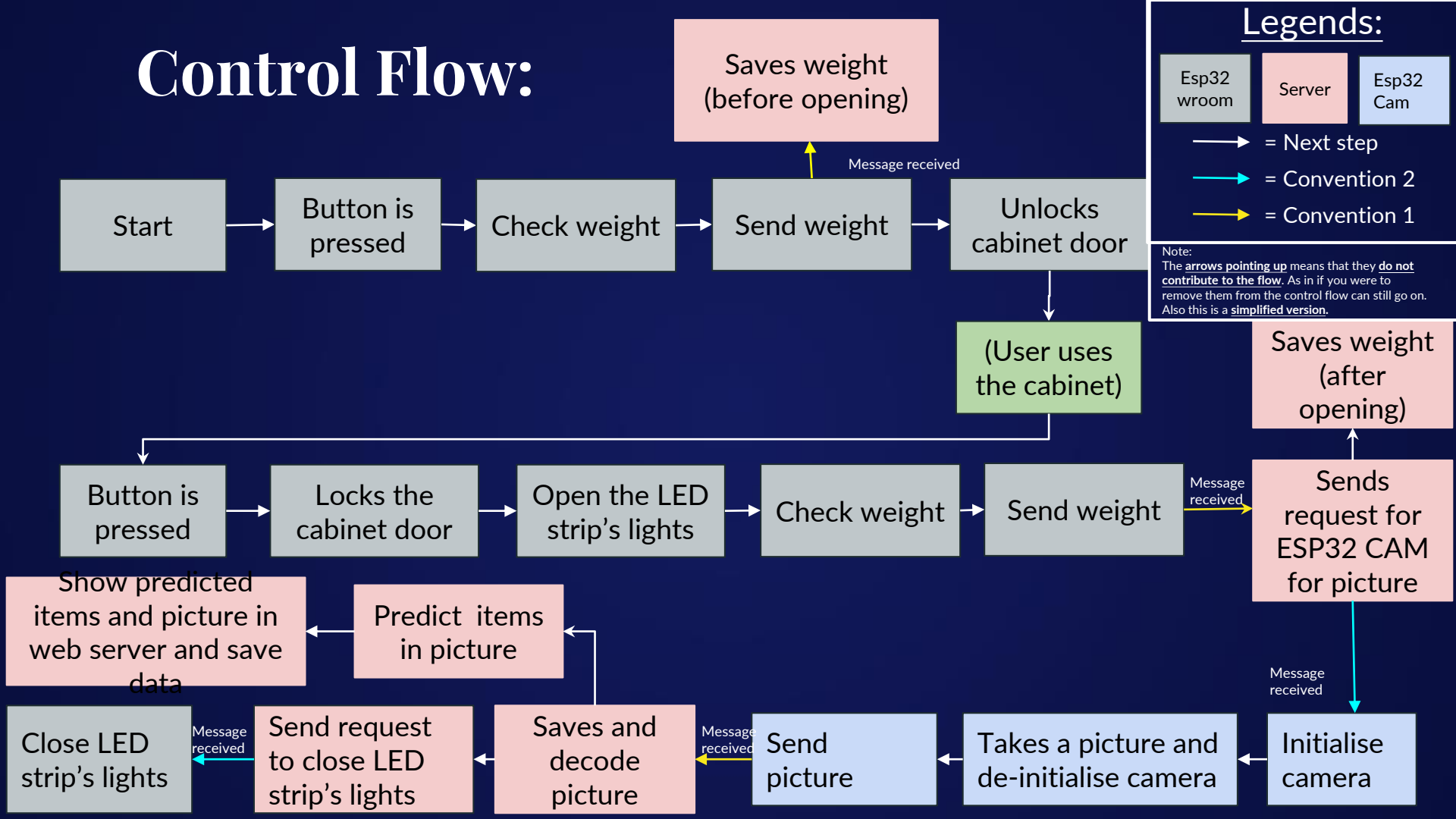
Convention 2

- Send the information from the transmitter to receiver
  - In the cabinet's case usually information are 4 characters long



- Only using the server to ESP client (more details in assignment)

Control Flow:



ESP32 CAM on Start

Initialise socket

```
client_socket = socket.socket()
client_socket.connect((SERVER_IP, SERVER_PORT))
```

Makes a socket object with the address family of AF_INET and of SOCK_STREAM socket type. Then connect to server

```
while True:
    msg = client_socket.recv(35)
    print(msg)
    if b"id" in msg:
        break
```

Wait for server to recognise that a client has been connected.

- Over here the client waits for the request for id from server

```
id_name = b"new~" + name.encode('utf-8')
client_socket.send(id_name)
print("confirm")
while b"start" not in client_socket.recv(32)
    continue
```

Sends the id it wants to be called on in the server side.

- Then wait until the server finishes registering the client
- More can be seen in the assignment

Repeat until we are connected

```
while sta_if.isconnected() != True:
```

Initialise Wifi

```
sta_if = network.WLAN(network.STA_IF)
sta_if.active(True)
```

Make WLAN network interface that connects to Wifi's Access points and activate the station interface

```
all_wifi = sta_if.scan()
```

Scans for all available access points

```
for wifi in all_wifi:
    if WIFI_SSID in wifi[0]:
```

Check if from all the access points they have the same ssid as the one we want to connect

```
sta_if.connect(WIFI_SSID, WIFI_PASS)
```

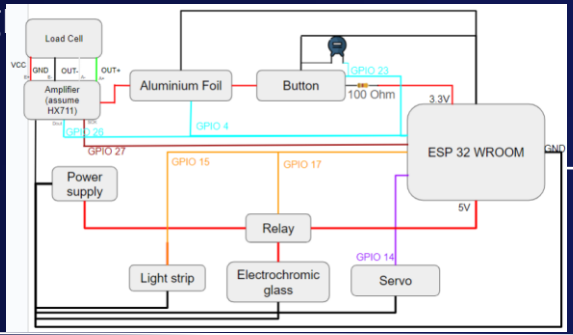
Connect to the wifi's access point of the SSID we specified

- Ensure that we will always connect to wifi if the access point is available

ESP32 Wroom on Start

Start

Initialise the pins according to the diag




```
# sensor and actuator initialisers
weight_sens = HX711(dout=26, pd_sck=27) # PD_SCK = digital out, Dout = digital in
weight_sens.set_scale(10)
weight_sens.tare()
motor = PWM(Pin(14) , freq = 50)
btn = Pin(23 , Pin.IN) # external pullup, does work
led_pin = Pin(15 , Pin.OUT)
led_strip = neopixel.NeoPixel(led_pin, NO_OF_LED)
glass = Pin(17, Pin.OUT)
touch_pin = TouchPad(Pin(4))
```

Initialise the weight sensor readings

```
m , c = cali_weight(weight_sens , EMPTY_WEIGHT , CALI_WEIGHT)
```

- On start, weigh the cabinet when its empty(128g) and when it has a predetermined weight(481g)
  - It reads the load cell for 50 times and give you the average of the weight

```
if start_cali == "e":
    init_weight = float(weight_sens.read_average(times = 50))
if start_cali == "c":
    const_weight = float(weight_sens.read_average(times = 50))
```

- Assuming the weight and load cell readings are linearly proportional
  - Find the gradient and y-intercept

```
m = (CALI_WEIGHT - EMPTY_WEIGHT)/(const_weight - init_weight)
c = init weight
```

Similarly to ESP32 CAM, initialise for socket and wifi

Button is pressed

Button is pressed

```
btn.irq(trigger=Pin.IRQ_FALLING, handler=handle_interrupt)
```

- I set it as falling so even if you hold the button there will only be counted as 1
- Also, for Ux it feels better to press and immediately get to see results
  - Instead of lifting your finger from the button

For debounce and resistance against noise coupling

```
def handle_interrupt(pin):
    '''
        FYI:
            - The button requires a capacitor because it is very inconsistent without it
                - As in it will affect the chromatic glass's gpio
    '''
    global door_open , send_info, lastDebounce
    current = time.ticks_ms()
    if time.ticks_diff(current, lastDebounce) <= 1500:
        print("Exiting interrupt")
        return
    time.sleep_ms(50)  # Short delay to filter out noise
    lastDebounce = current
    if btn.value() == 0:  # Check if the button is still pressed. for debounce
        door_open = not door_open
        send_info = True
        print(f"button {door_open}")
```

Acts as flags for the conditions in the main code

```
if send_info == True and door_open:
```

```
elif send_info and not door_open:
```

Check weight

*reads 50 times of the load cell and finds the average

Send weight

Convention 1

```
weight = float(weight_sens.read_average(times = 50)) * m / 10
print(f"weight {weight}")
weight = str(weight).encode("utf-8")
client_socket.send(b"size-{}".format(len(weight))) # size decl
while send_msg == False:
    continue
```

```
req_server = client_socket.recv(35)
```

```
if b"send" in req_server:
    send_msg = True
```

```
weight = f"{state}-".encode("utf-8") + weight # declare what s
client_socket.send(weight)
send_msg = False
```

Saves weight (before opening)

```
elif b"bfr" in data[:4]:
    print(f"before {data_str}")
    before_weight = data_str
```

Unlocks cabinet door

```
motor.duty(80)# open
```

Locks the cabinet door

```
motor.duty(30) # close
```

Open the LED strip's lights

Sets all LEDs to (r = 255,g = 255,b = 255)

```
led_rgb_setter(led_strip  , NO_OF_LED , 255 , 255 ,255)
```

Check weight

*reads 50 times of the load cell and finds the average

```
weight = float(weight_sens.read_average(times = 50)) * m / 10
print(f"weight {weight}")
weight = str(weight).encode("utf-8")
client_socket.send(b"size-{}".format(len(weight))) # size decl
while send_msg == False:
    continue
```

Send weight

Convention 1

```
req_server = client_socket.recv(35)

if b"send" in req_server:
    send_msg = True
weight = f"{state}-".encode("utf-8") + weight # declare what s
client_socket.send(weight)
send_msg = False
```

Sends request for ESP32 CAM for picture

```
if b"aft" in data[:4]:
    print(f"after {data_str}")

    send_img_req()
```

This is done so that nothing can be changed before I read and take the picture

Sends request for ESP32 CAM for picture

Convention 2

Finds all the clients known id called img

```
def send_img_req():
    for keys in client_to_send.keys():
        print(f"stuff {client_to_send[keys][1]}")
        if b"img" == client_to_send[keys][1]:
            print(f"sending req to {client_to_send[keys][0]}")
            client_to_send[keys][0].send(b"img")  # Send image request to client
    return "sent"  # Return sent
```

```
req_server = client_socket.recv(35)
print("requested")
if req_server == b"img":
```

Saves weight (after opening)

```
if b"aft" in data[:4]:
    print(f"after {data_str}")
    curr_weight = data_str
```

Initialise camera

Takes a picture and de-initialise camera

```
try:
    camera.init(0, format=camera.JPEG
    pic_in_bytes = camera.capture()
    print(len(pic_in_bytes))
    size = len(pic_in_bytes)
    return pic_in_bytes , size
except Exception as e:
    print("error")
finally:
    camera.deinit() # de initialise a
```

- I used finally in try and except because in any case it will always deinit the camera

Send picture

```
client_socket.send(b"size-{}".format(len(msg)))
while b"send" not in client_socket.recv(32): #
    continue
client_socket.send(msg) # sends message
return True
```

Saves and decode picture

```
img = Image.open(io.BytesIO(data))
pic_name = '1.png'
pic_name = "/Users/Sanjeev/Desktop/IOT sys project/static/images/" + pic_name
```

```
img.save(pic_name)
```

We decided to overwrite our pictures after every receive

Convention 2

Send request to close LED strip's lights

```
for keys in client_to_send.keys():
    print(f"stuff {client_to_send[keys][1]}")
    if b"cab" == client_to_send[keys][1]:
        print(f"sending req to {client_to_send[keys][0]}")
        client_to_send[keys][0].send(b"lof")  # Send LED off request to client
return "sent cab"  # Return sent cab
```

Finds all the clients known id called img

Close LED strip's lights

```
req_server = client_socket.recv(35)
print(f"requested {req_server}")
if req_server == b"lof": # request to close lights
    led_rgb_setter(led_strip  , NO_OF_LED , 0 , 0 ,0)
```

Sets all LEDs to (r = 255,g = 255,b = 255)

Predict items in picture

Create Yolo object

```
model = YOLO(model_path)  # Load the YOLO model
predict_results = model.predict(image, imgsz = 640 , conf = 0.6 , save = True)
```

For items that are predicted as x item for 60% or more confidence are outputted.
The image size allowed is only 640.
The images will then be annotated and saved in a runs/detect/predict_

- Where _ is the amount of folders of predict there are
- e.g. under runs/detect/
  - predict
  - predict2
  - predict3

```
for r in predict_results:
  boxes = r.boxes  # Boxes object for bbox outputs
```

```
predicted_food = boxes.cls.tolist()
```

Get Each item's boxes object.
Find its classification and put it into a list from the boxes object

```
if predicted_food[i] == 0:
  all_items["Nutella"] += 1
elif predicted_food[i] == 1:
  all_items["Pepsi"] += 1
elif predicted_food[i] == 2:
  all items["Pringles"] += 1
```

I set it as:

- 0: nutella
- 1: pepsi
- 2: pringles

As yolo's convention for labelling is numbers

Show predicted items and picture in web server and save data

Make all items into the Item object

Add the items predicted into the "Item" table

Save changes

```
with app.app_context():
    Item.query.delete()
    for i in item:
        new_item = Item(name= i,quantity = item[i])
        db.session.add(new_item)
        db.session.commit()
```

Show predicted items and picture in web server and save data

To show image on the webpage

```
<div class="img-container">
    <img id="img" width="160" height="160">
    <img src="{{ url_for('static', filename='images/1.png') }}" alt=""width="160" height="160">
</div>
```

Update the items in the html using jinja

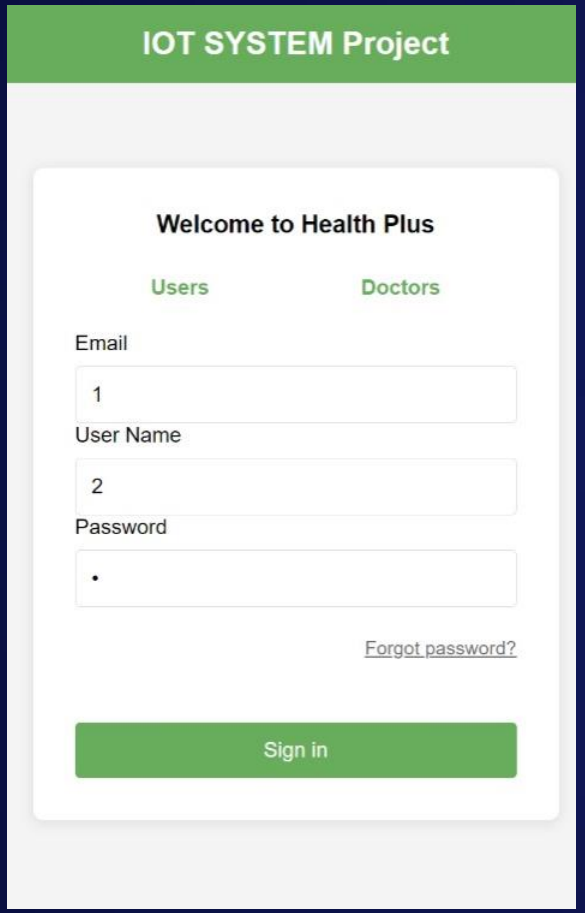
```
@app.route("/cabinet")
def cabinet():
    items = Item.query.all()  # Query all items
    return render_template('cabinet.html', items=items)
```

Query the whole Item table and put it into jinja's arguments

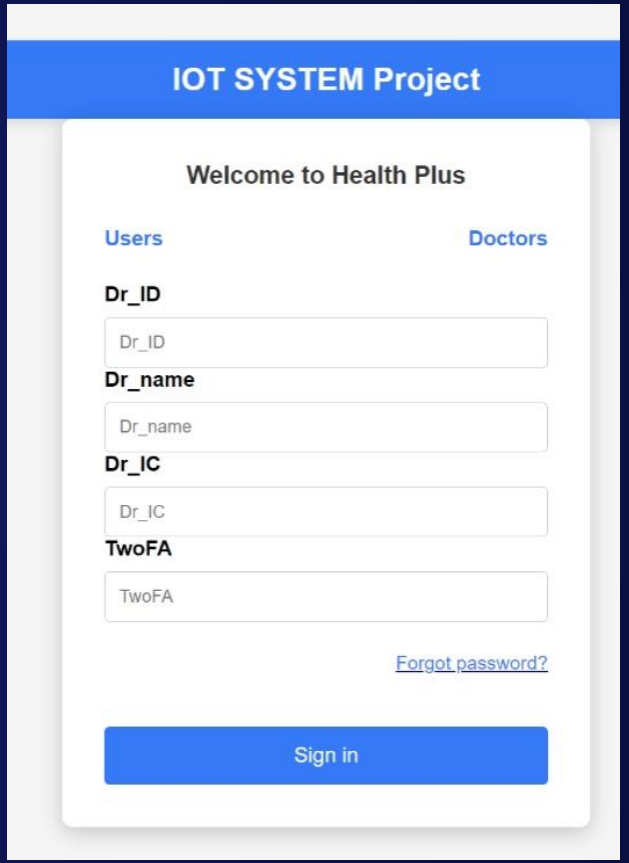
Loops through all the items and find its item name and quantity

```
<ul id="cabinet-items-list">
    {% for item in items %}
        <li>{{ item.name }}: {{ item.quantity }}</li>
    {% endfor %}
</ul>
```

Dashboard/Website


IOT SYSTEM Project
Welcome to Health Plus
Users
Doctors
Email
1
User Name
2
Password
•
Forgot password?
Sign in

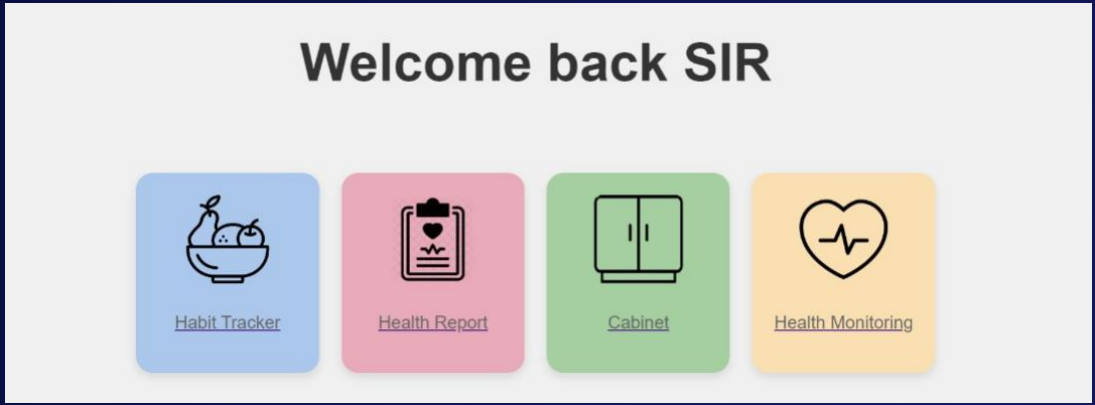
LOGIN PAGE


IOT SYSTEM Project
Welcome to Health Plus
Users
Doctors
Dr_ID
Dr_ID
Dr_name
Dr_name
Dr_IC
Dr_IC
TwoFA
TwoFA
Forgot password?
Sign in

Data Sheet

Time	Heart Rate (HR)	Temperature (Temp)
11 Jul 08:49:58	44.00	23.11
11 Jul 08:50:28	44.00	23.11
11 Jul 08:51:03	28.00	23.11
11 Jul 08:51:33	52.00	23.11
11 Jul 08:52:08	48.00	23.11
11 Jul 08:52:38	48.00	23.11
11 Jul 08:53:13	48.00	23.11
11 Jul 08:53:48	56.00	23.11
11 Jul 09:04:11	88.00	23.11
11 Jul 09:04:41	60.00	23.11
11 Jul 09:05:11	152.00	23.11
11 Jul 11:21:01	8.00	23.11

Home page



This is the main page where it links to the the Health monitor, Cabinet, Health Report and Habit tracker to get more info on each category

Habit Tracker

The habit tracker analyses and track the nutrients you have eaten based on the data from the cabinet and the smart watch

Health Report

The health report helps you view your health report from the hospital and allows you to send health comments to the doctor

Cabinet

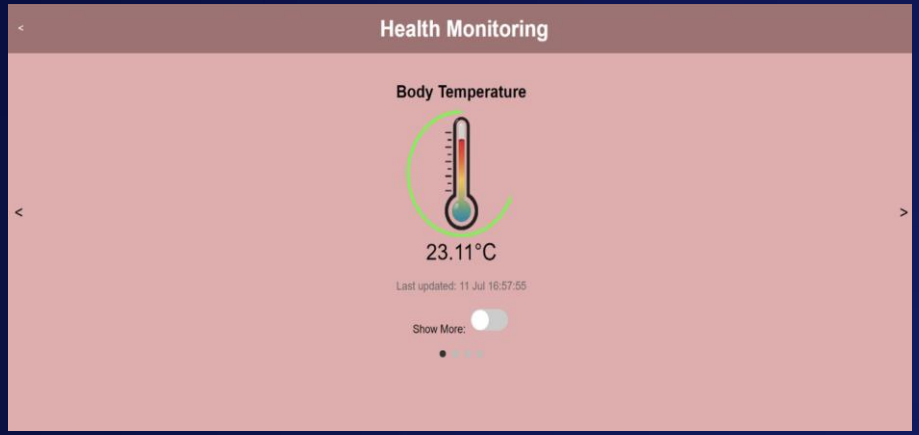
The Cabinet allows you to view the items in the cabinet via your phone such that it would allow easier logistics as you do not need to remember the items in the cabinet

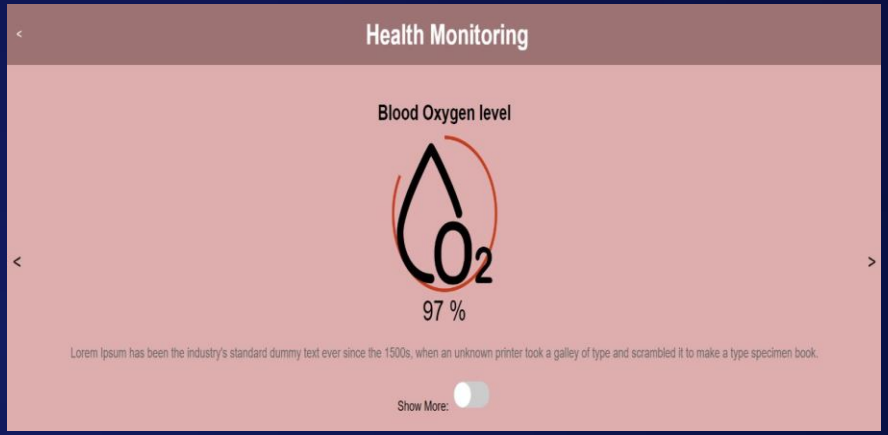
Health monitoring

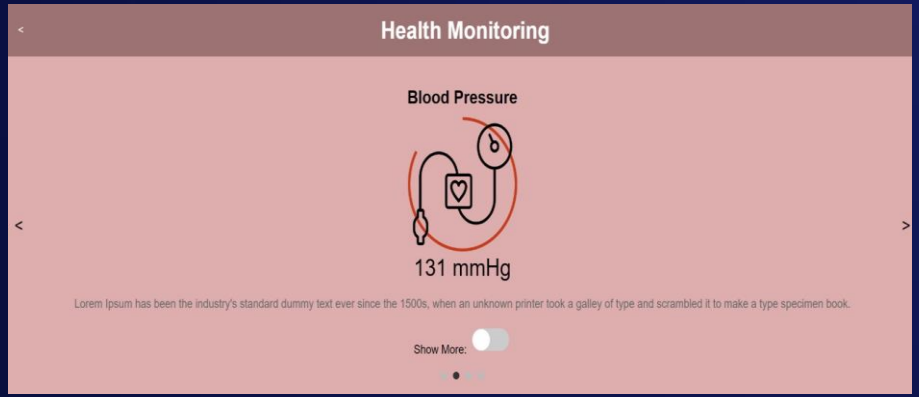
The Health monitoring helps track the smart watch users health data like heart rate, blood glucose,blood oxygen meter, and body temperature in real time and also within a range of a week

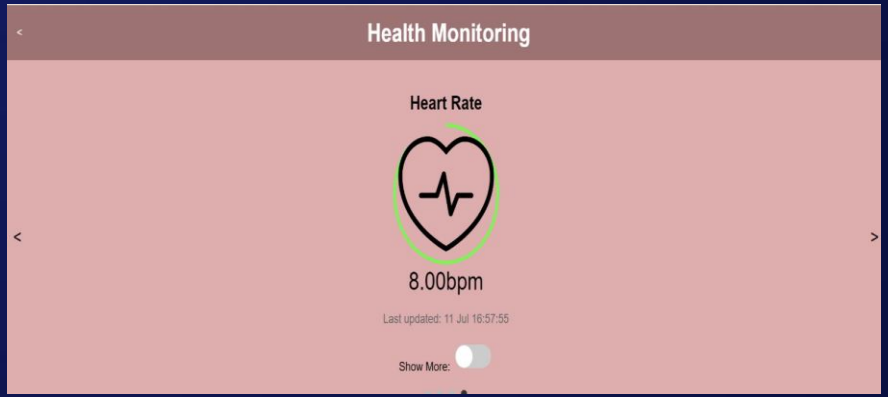
Health Monitoring page

Tracks the real time health data and updates the round progress bar around the icons to show a sense of healthiness which is based on a range for a healthy person



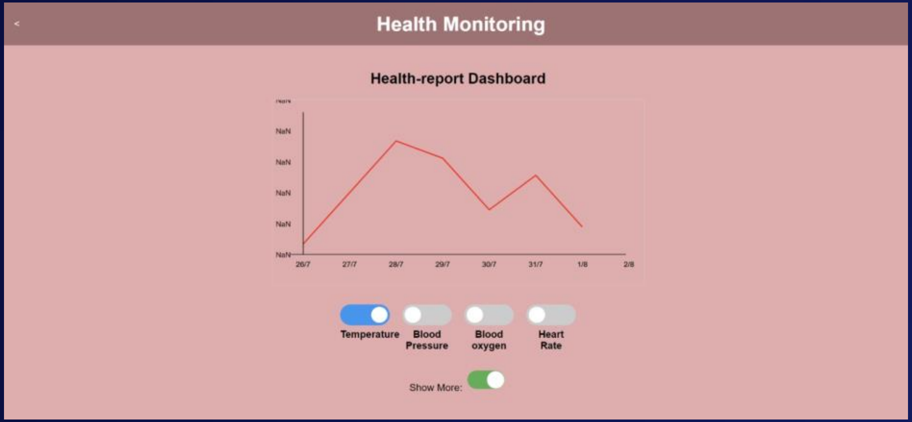


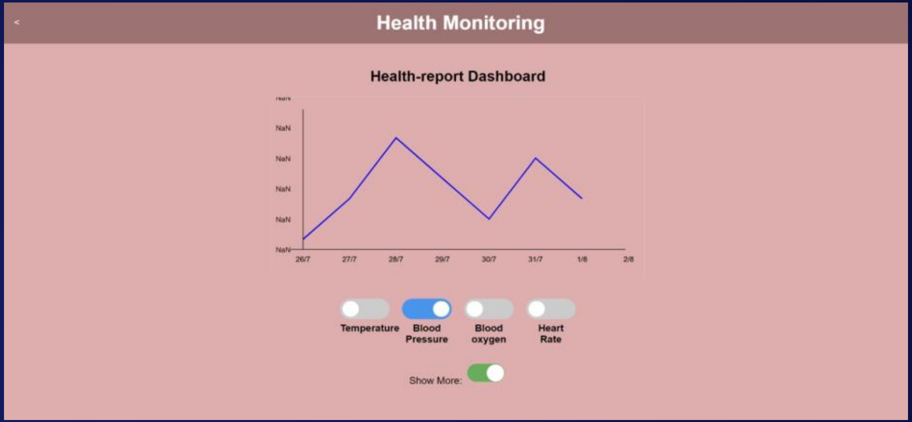


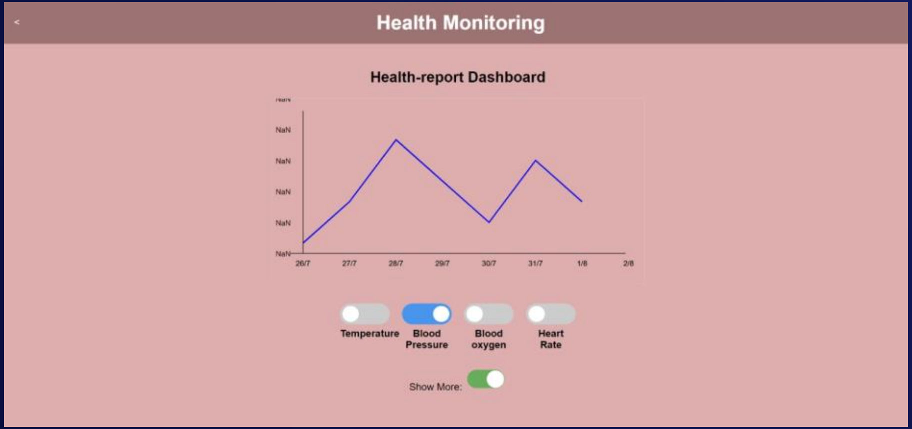


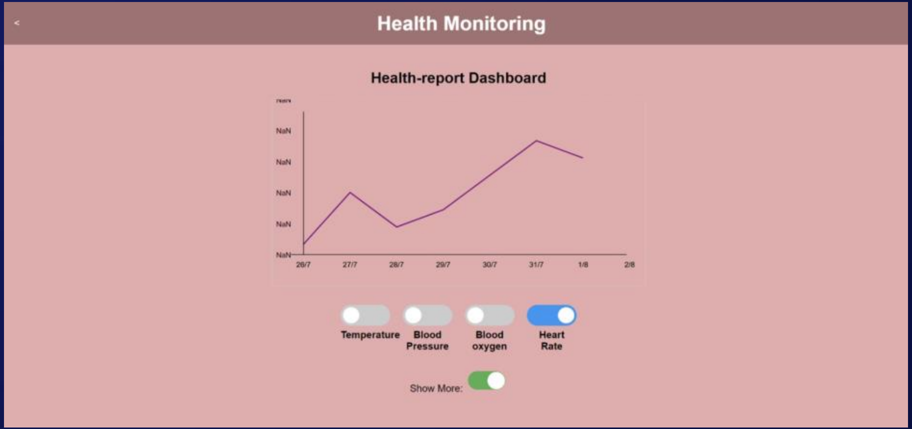
Health Monitoring data

Displayed health data from over the past 7 days (week) so that it is easier to detect any abnormalities in the health data.



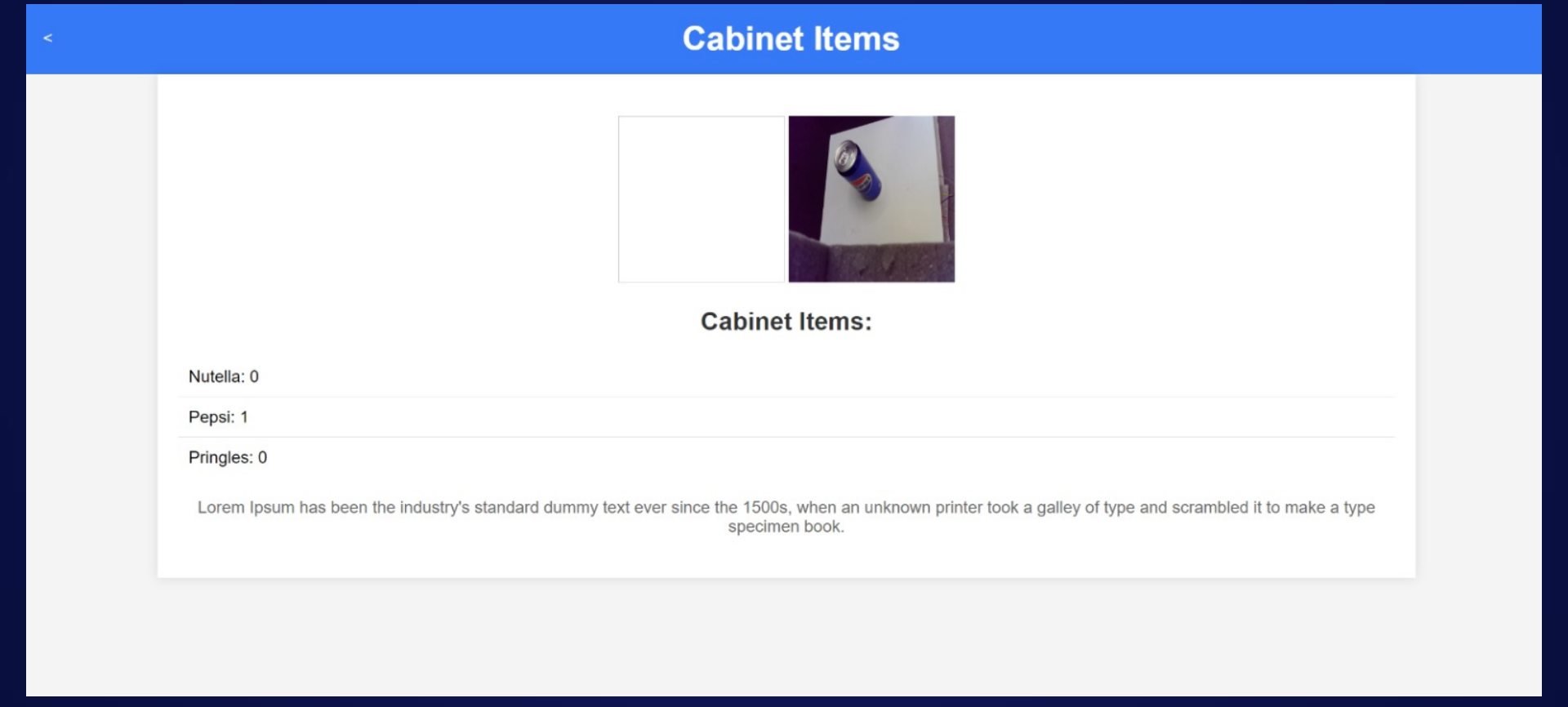






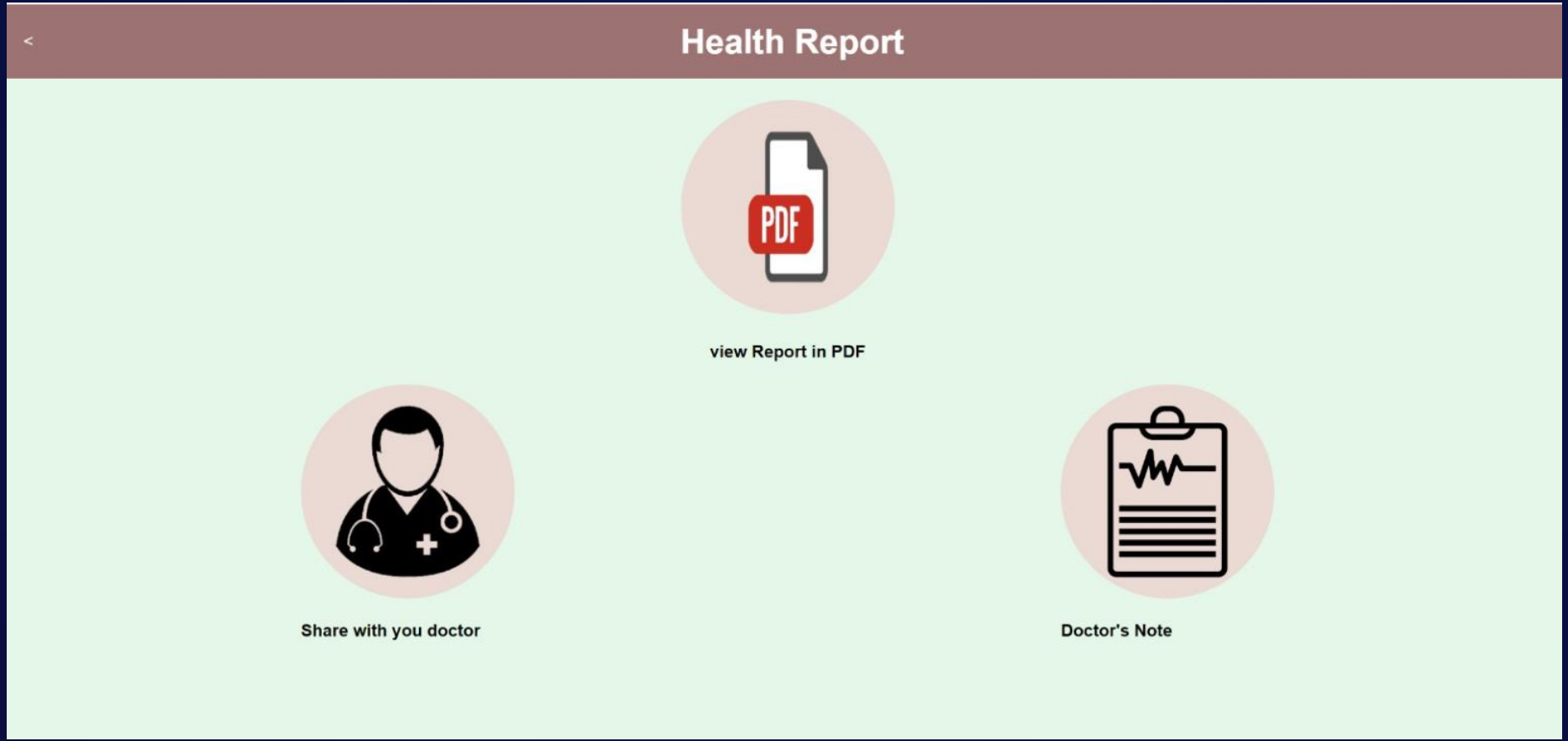
Cabinet

Displayed a the latest photo of inside the cabinet from the last time the cabinet was open



Health Report

helps you view your health report from the hospital and allows you to send health comments to the doctor



Health Report viewer

This page shows your health report stats all your medical report details and the comments and suggestions from the doctor to improve your daily lifestyle habits

Health Report

Blood Test Results

Total Cholesterol	1
Triglyceride	2
HDL-C	1
LDL	2
Chol/HDL	1
Bilirubin	2
ALT	1
AST	2
ALP	1
Golbulin	2
Protein	1
Albumin	21
Creatinine	1
Potassium	2
eGFR	3
HbA1c	4

Doctor's Advice

Cardiovascular: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Glucose: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Diet: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Exercise: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Mental: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Respiratory: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Digestive: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Bone Health: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Immunizations: Listen to your body's whispers, not its shouts. Daily habits like regular sleep, healthy meals, and movement are the foundation of health. Don't wait for illness to strike. Let's build a personalized plan together. Together, we can cultivate a vibrant you!

Download as PDF

Share with doctor

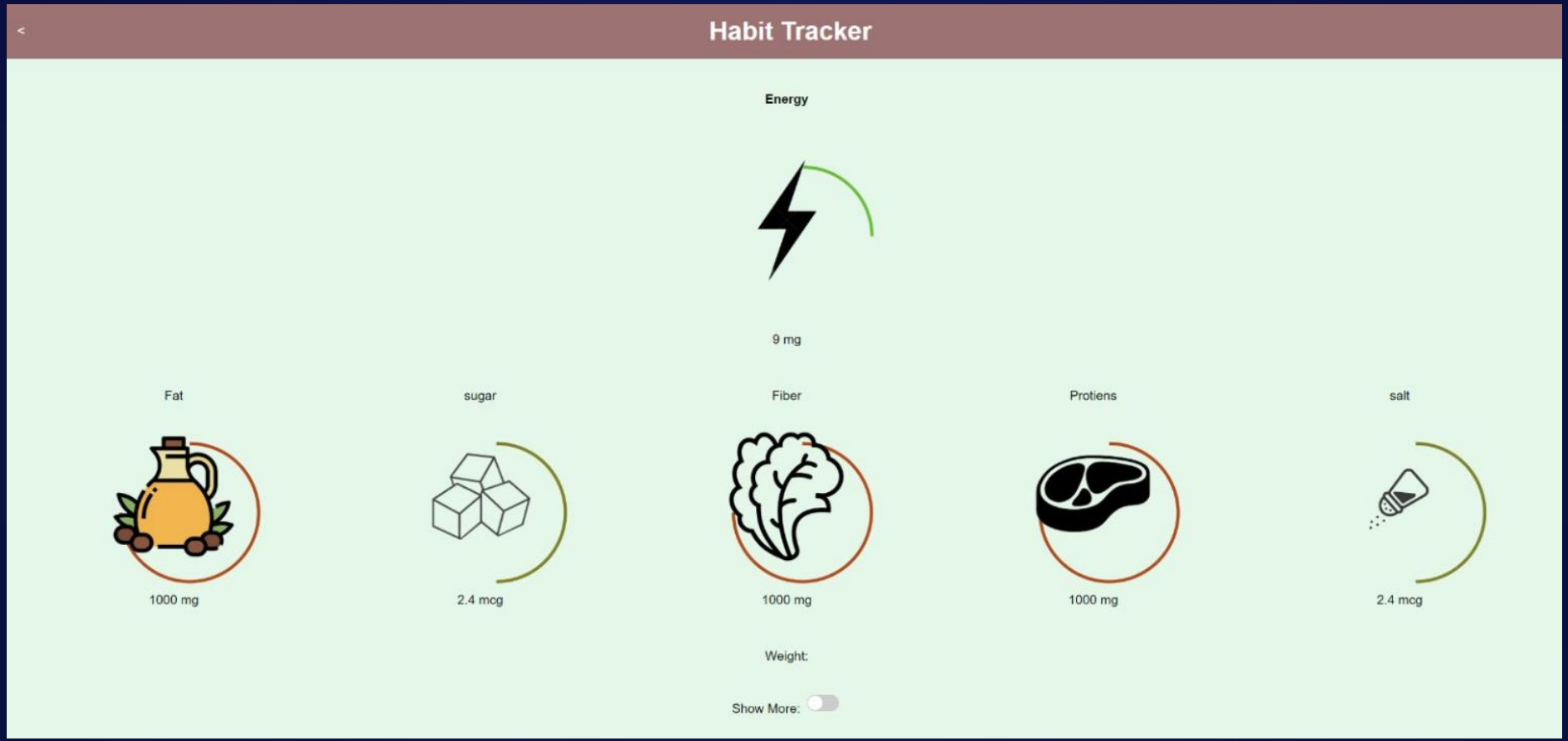
Allows the user to comment on their health so that the doctor can read it and get a better understanding on the patient and diagnose them better

Doctor’s note

Allows the users to view comments and suggestions from the doctor to improve your daily lifestyle habits

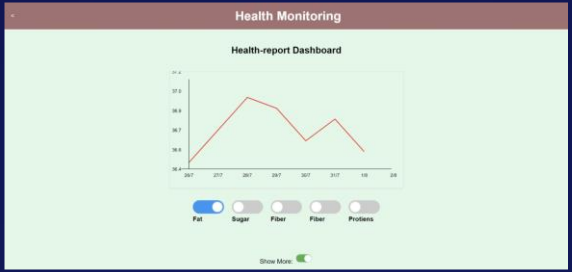
Health Tracker data

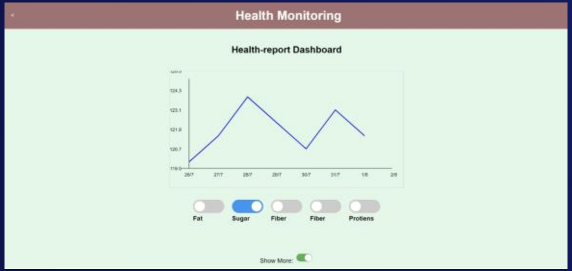
Displays the macro nutrients that the user has intake and loss based on the cabinet and smart watch.

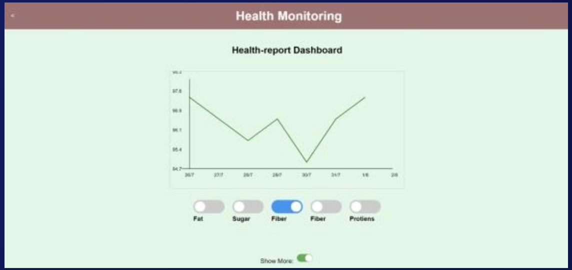


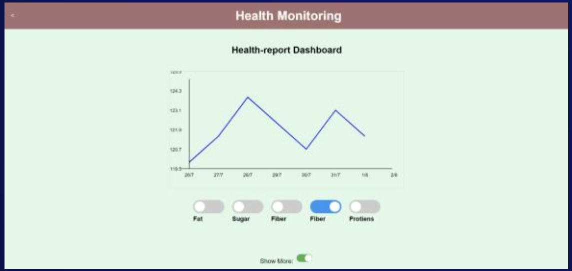
Health Tracker data

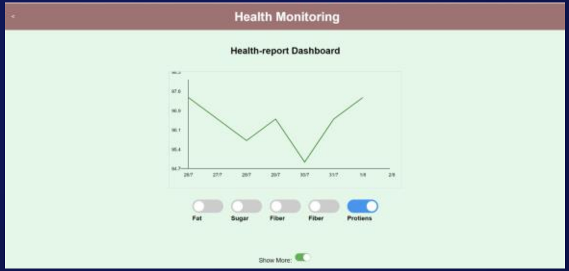
Displayed health data from over the past 7 days (week) so that it is easier to detect any abnormalities in the health data.











07

Problem & Solution

Problem & solution (Hardware)

Problem	Solution
ESP32 Cam rarely is able to take good quality photos for the ML model to predict	- Don't use the camera for more than 1 hour - If the camera starts to give error photos press the "RST" or unplug and plug

RST

Error photos

Shell

```
error
error
error
error
error
error
error
error
```

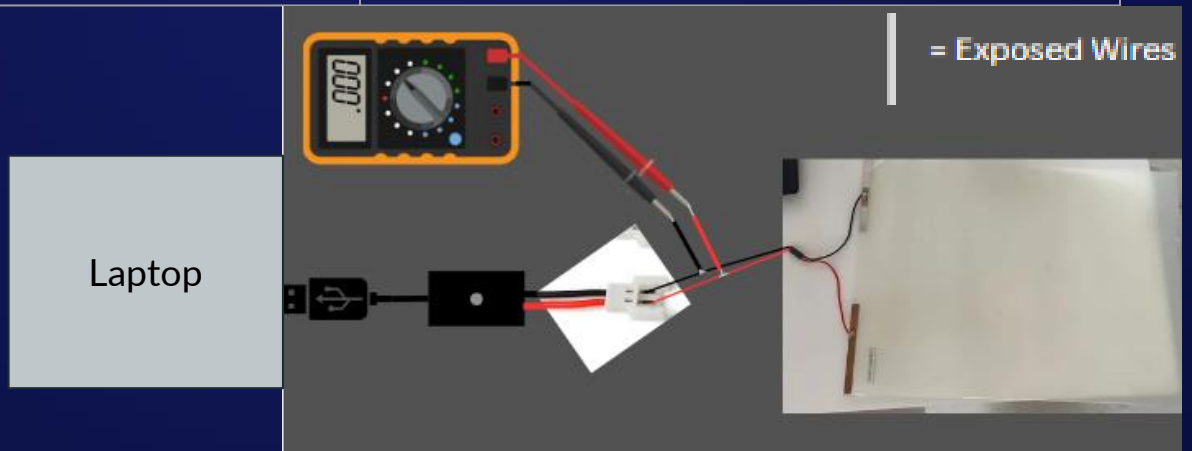
Good photo

Problem & solution (Hardware)

Problem	Solution
The connectors keeps coming loose and loosing connection for the electrochromic glass	We decided to wire wrap the connection cables
Pins of the esp32 causes noise coupling and causes abnormal data readings for the button	We added capacitors to the button’s pin to the ground to reduce the noise and change the location as far from other pins

Problem & solution (Hardware)

Problem	Solution
Did not know the voltage needed to supply the electrochromic glass. (The manufacturer did not give any datasheet)	Soldered two pieces of wires to be able to measure the voltage in parallel. Then Connect these wires to the jumper and connector



Problem & solution (Hardware)

Problem	Solution
Electrochromic glass requires high voltage (around 20 - 34V) and low current	Used the USB connector(that has a transformer) and connect it via a relay to supply or leave the electrochromic glass open
Lack of brightness for the camera	We experimented with led strips, UV light strips and in the end we used a ceiling light with around 30k lumens
The box was to bright after light addition and the white colour was reflecting the light causing the image to be over exposed	We added black foam to the sides to reduce the amount of light bouncing.

Problem & solution (Software)

Problem	Solution
There was no readily available library for the pulse sensor	We had to create a custom library to run the pulse sensor
The Pulse sensor data being received was to noisy and unreadable	We added a Kalman filter to reduce the amount of noise
Having 2 threads that receive sockets. This makes the order of where the message gets received is ambiguous	Put all the receive into 1 socket receive

Problem & solution (Software)

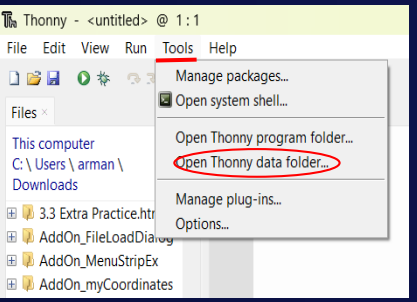
Problem	Solution
The button was unreliable due to bouncing	Added a debouncing code and extra delay due to the electrochromic glass
The frame buffer will never stay constant because there are many users sending info to the server	The ESP32 clients usually declare their size of message before sending and if any messages are received before then, they will use the largest frame buffer - Hence,the frame buffer will always be bigger than any message
It is hard to identify and call the whole remote address to send info to a esp32 client	Associate the given id(from the esp client on start) with its remote address on the server. So it is easier for the programmer to send info to a client.

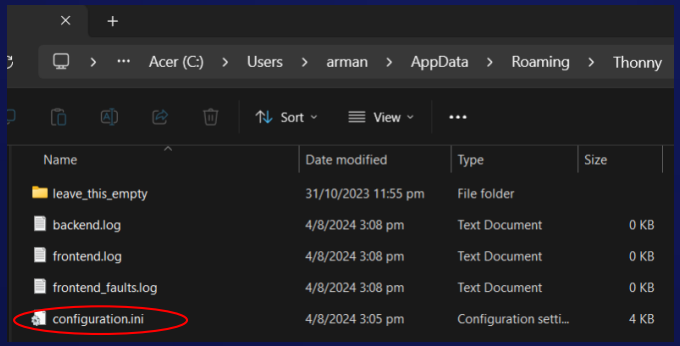
Problem & solution (Software)

Problem	Solution
For the CV training, the pictures were taken from many sources. Hence, relabeling and splitting of images is required	I made a script to rename all the labels of nutella(as it had a different label to the other dataset I have) and split the images into training, testing and validation
The model was not able to predict properly predict	Added some extra pictures taken from the esp32 cam as training,testing and validation dataset

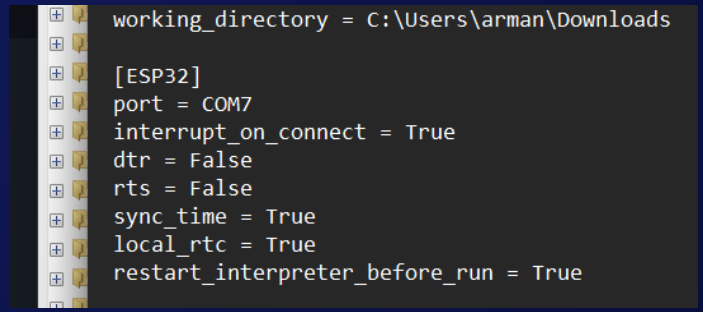
Problem & solution (Software)

Problem	Solution
Serial monitor of Thonny was not working For Esp32 Cam	Set the following parameters for configuration.ini





Configuration.ini



Problem & solution (Website & server)

Problem	Solution
Adafruit is very restrictive in the amount of features able to add and not enough feature	We decided to create our own website and Server to communicate between the esp32’s and website using knowledge of html, css, js & python

08

Reference & Conclusion

References

- https://youtu.be/8A4dqoGL62E?si=o7QooOgwWbd1o6Av
- https://youtu.be/436VDF-rk4w?si=HLJeLFKR8mP-bj8Y
- https://youtu.be/Kg-sxVmCt5Q?si=pJ8aw7vPxOXJIFL6
- https://youtu.be/xZF6zWLz-vY?si=vn6EXkiPK49Wyo5H
- https://youtu.be/TDgM8eMTpIw?si=Jgo6hKibeRnFMYET
- https://lastminuteengineers.com/pulse-sensor-arduino-tutorial/
- https://randomnerdtutorials.com/esp32-touch-pins-arduino-ide/

Troubleshooting:

- https://github.com/thonny/thonny/issues/2845
- https://blog.csdn.net/zhusongziye/article/details/128176230
- https://github.com/thonny/thonny/issues/2282

Documentation ,libraries and firmware

- https://github.com/lemariva/micropython-camera-driver
- https://docs.python.org/3/library/socket.html
- https://docs.ultralytics.com/models/yolov9/#usage-examples
- https://docs.micropython.org/en/latest/index.html

Dataset:

- https://drive.google.com/drive/folders/1Y50k5xA9a6E_1hBGsiXxJK3iCsmsFKud?usp=sharing